Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tadej Hiti

Proceduralno generiranje vsebine za računalniško igro v kombinaciji s stilom low poly

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Narvika Bovcon

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomske naloge predstavite nizkopoligonski oblikovni slog in princip postopkovnega grajenja vsebine za računalniško igro. Pojasnite, zakaj ju uporabljamo. Predstavite postopek izdelave serije raznolikih nizkopoligonskih modelov in opišite razvoj programske opreme za grajenje mesta, s pomočjo katerih ste izdelali vsebino za računalniško igro.

Zahvaljujem se izr. prof. dr. Narviki Bovcon za hitro dosegljivost in pomoč pri izdelavi diplomskega dela. Posebej se zahvaljujem tudi družinama in partnerici za podporo med študijem.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Tadej Hiti, z vpisno številko 63130064, sem avtor diplomskega dela z naslovom:

Proceduralno generiranje vsebine za računalniško igro v kombinaciji s stilom low poly (angl. Low poly models and procedural generating of content for computer games).

S svojim podpisom zagotavljam, da:

- sem diplomsko nalogo izdelal samostojno pod mentorstvom izr. prof. dr. Narvike Bovcon,
- da je tiskana oblika pisnega diplomskega dela istovetna elektronski obliki pisnega diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 1. september 2017 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
PCG	procedural content generation	proceduralna generacija vsebine
AAA	triple-A	trojni-A
RAM	random access memory	primarni računalniški pomnilnik
macOS	Macintosh operating system	operacijski sistem Macintosh
GPE	graphical procesing unit	grafično procesna enota
API	application programming interface	aplikacijski programski vmesnik

Povzetek

Naslov: Proceduralno generiranje vsebine za računalniško igro v kombinaciji s stilom low poly

Avtor: Tadej Hiti

Industrija iger je zaradi nenehnih tehnoloških premikov v strojni in programski opremi eno izmed najhitreje razvijajočih se področij računalništva, s čimer je povezana tudi velika konkurenca na trgu. Proizvajalci stremijo k izdajanju vedno kompleksnejših, grafično in vsebinsko bogatejših in strojno zahtevnejših vsebin, pri tem pa povečujejo razvojni čas in s tem posledično tudi ostale stroške produkcije. Proizvajalci iščejo nove poti in rešitve, ki bi jim omogočile izdajo končnih verzij produktov z minimalnimi stroški razvoja brez izgub na kakovosti. Ena izmed popularnejših rešitev je uporaba tehnik proceduralnega generiranja vsebine. Slednje združeno s poenostavljenim izgledom, imenovanim „low poly“, zagotavlja veliko več kot zgolj zmanjšanje razvojnega časa in ostalih stroškov produkcije: razvijalcem ponuja tudi zelo hiter in bogat način predstavitve željene vsebine. Namen diplomske naloge je raziskati in predstaviti oblikovani stil low poly ter princip proceduralnega generiranja vsebine, pridobljeno znanje pa združiti in uporabiti pri razvoju programske opreme, ki uporablja proceduralno generiranje vsebine za prikaz mesta, čigar modeli so oblikovani v stilu low poly.

Ključne besede: postopkovno generiranje, low poly, računalniške igre, modeliranje, upodabljanje.

Abstract

Title: Low poly models and procedural generating of content for computer games

Author: Tadej Hiti

Because of the never ending technological breakthroughs of software and hardware limitations in computer science the industry of games is one of the most expanding markets, flooded with astonishing number of games and high competition. Because of that, developers are constantly striving towards releasing more complex video games with astonishing amount of detail in it's content and graphics, producing ever so rich and hardware demanding products, meanwhile pushing the budget costs above acceptable limit. Because of this constraints the developers are seeking towards better, faster ways to produce their content at the acceptable costs without the loss of quality. One of the popular ways to achieve such goals is to use procedural content generation when making content of their games. Merging the following concept with the use of a minimalistic low poly art style to represent their assets, allows developers to achieve desired goals with rich and quickly produced content. The purpose of this diploma thesis is to research and present low poly art style combined with procedural content generation and use obtained knowledge to create a software that will use procedural content generation in combination with low poly assets to produce ever different city representations.

Keywords: procedural content generation, low poly, computer games, modeling, shading.

Poglavje 1

Uvod

Industrija iger je zaradi nenehnih tehnoloških premikov v strojni in programski opremi eno izmed najhitreje razvijajočih se področij računalništva, s čimer je povezana tudi velika konkurenca na trgu. Zaradi želje po vedno boljših, močnejših, hitrejših računalniških sistemih in računalniških komponentah rastejo tudi uporabnikove zahteve po bogatejši vsebini, ki jo ponujajo proizvajalci. S tem uporabniki diktirajo tempo trga in ustvarjajo višje povpraševanje, kar povzroča večjo konkurenčnost podjetij, ki izdajajo zahtevano vsebino. Zavoljo izpolnjevanja vedno višjih zahtev uporabnikov in zaradi visoke zasičenosti trga s podobnimi produkti proizvajalci računalniških iger stremijo k izdajanju vedno kompleksnejših, vsebinsko bogatejših in strojno zahtevnejših iger ter s tem povečujejo razvojne stroške, potrebne za izdajo končnih verzij teh produktov. Zaradi povečanega časa razvoja se posledično povečajo tudi stroški produkcije, za kar pa proizvajalci venomer iščejo nove načine in poti po minimizaciji le teh. Ena izmed popularnejših možnosti je proceduralna generacija vsebine v igrah. Je tehnika, ki s človeško pomočjo ustvarja željeno vsebino algoritmično oziroma proceduralno. Proceduralna generacija vsebine tako na primer znatno pomaga manjšim ekipam z veliko manj človeške moči (v primerjavi s tistimi z visokimi letnimi donosi in večjim številom zaposlenih) izdati konkurenčen produkt z višjo stopnjo grafične raznolikosti in svežine ter nižjim potrebnim razvojnim časom. Prvo-

tni proceduralni algoritmi v osemdesetih letih prejšnjega tisočletja so nastali predvsem zaradi potrebe zmanjšanja velikosti računalniških iger [21]. Proizvajalci se vedno bolj poslužujejo ustvarjanja vsebine v tako imenovanem low poly oblikovnem slogu. Gre za uporabo minimalnega števila poligonov v 3D modeliranju za minimalno in abstraktno ponazoritev željenega pomena. V sklopu diplomske naloge si bomo podrobneje ogledali področja oblikovnega stila low poly in pomen proceduralnega generiranja vsebine, v nadaljevanju pa oba pristopa združili v končni produkt, ki bo predstavljen kot razvoj programske opreme, ki uporablja proceduralno generiranje vsebine za prikaz mesta, čigar 3D modeli bodo oblikovani v geometriji low poly. Predstavili bomo uporabljene tehnologije, podrobno pregledali, kako smo v stilu low poly oblikovali dane modele, ki predstavljajo stavbe, cestne odseke, vegetacijo danega mesta in ostale podrobnosti urbanega okolja. Nazadnje bomo predstavili, kako smo s pristopi uporabe proceduralnega generiranja vsebine ustvarili mesto, ki bo na podlagi različnih vrednosti vhodnih parametrov (višina stavb, gostota poseljenosti mesta, pas obsega stolpnic, stopnja gostosti vegetacije, itd...) reprezentiralo različne oblike slednjega. Pri vsem tem pa nam bo cilj predstaviti vso načrtovano vsebino v aplikaciji, ki teče v realnem času. Motivacija za izbrano diplomsko temo je bilo moje davno zanimanje za „indie game development“ oziroma samostojno razvijanje iger. Za računalniške igre, ki jih razvijam, želim ustvariti tako grafično vsebino, da bom lahko z njo hkrati povečal zadovoljstvo uporabnikov in zmanjšal stroške razvoja.

Poglavje 2

Interaktivna računalniška grafika in low poly

Preden se dotaknemo opisovanja koncepta stila low poly grafike se najprej seznanimo z nekaj osnovnimi pojmi in koncepti računalniške grafike, ki bodo neukemu bralcu omogočile lažje razumevanje konceptov ter metodologij, opisanih v nadaljevanju tega poglavja.

2.1 Definicija

Računalniška grafika je zelo obsežen pojem področja računalništva, vendar ga lahko v nekaj besedah opišemo kot: vsa računalniško generirana slikovna informacija, ustvarjena s pomočjo specializiranih orodij strojne ter programske opreme za manipulacijo tovrstnega slikovnega materiala [6]. V prihajajočih poglavjih, ko bomo govorili o predstavitvi low poly grafike, se bomo navezovali na področje računalniške grafike, imenovane interaktivna računalniška grafika, oziroma natančneje 3D računalniška grafika.

Interakitvna računalniška grafika omogoča uporabnikov poseg ali interakcijo z računalniškim sistemom, ki prikazuje željeno manipulirano grafično informacijo.

3D računalniška grafika shranjuje, manipulira ter kasneje upodablja željeno geometrijsko informacijo v kartezičnem tridimenzionalnem prostoru kot dvodimenzionalni slikovni material na zaslonu danega računalniškega sistema. Informacijo v 3D računalniški grafiki imenujemo kar 3D geometrijski model ali krajše 3D model, predmet ali objekt, v nadaljevanju bomo zanj uporabljali ta imena brez predpone „3D“.

3D računalniška grafika ponuja več možnosti zapisov predstavitve modela [14]:

- Ploskve
  - poligonske mreže (angl. polygon meshes)
  - deljene ploskve (angl. subdivision surfaces)
  - parametrične ploskve (angl. parametric surfaces)
  - implicitne ploskve (angl. implicit surfaces)
- Krivulje
  - neenakomerne racionalne osnovne krivulje ali NURBS (angl. non-uniform rational basis spline)
- Neposredno zajeti podatki
  - voksli (angl. voxels)
  - oblaki točk (angl. point clouds)
  - globinske slike (angl. depth images)
- Polna telesa
  - osmiška drevesa (angl. octrees)
  - prostorsko binarna particijska drevesa ali BSP drevesa (angl. binary space partition trees)

– konstruktivna trdna geometrija ali CSG (angl. constructive solid geometry)

Večinoma so ploskve ali natančneje poligonske mreže najpopularnejša izbira za predstavitev modelov računalniške grafike, zato si oglejmo njihovo predstavitev v prihajajočem podpoglavju.

2.2 Poligonske mreže

Poligonske mreže so z razlogom najpopularnejša izbira za predstavitev predmetov v računalniški grafiki, saj so zmožne predstavitve skoraj vsake oblike predmeta, so enostavne za shranjevanje na računalniškem sistemu, ponujajo preprosto manipulacijo nad predmeti in so zmožne hitrega ter efektivnega procesiranja in izrisovanja na računalniški zaslon. Poligonska mreža je zbirka točk (angl. vertex), robov (angl. edge) in stranic ali poligonov (angl. face, polygon), ki definira obliko poliedričnega predmeta v 3D računalniški grafiki [13].
Poliedrični predmet ali polieder (angl. polyhedron, polyhedra) je zaprta poligonska mreža (tj. mreža, definirana s končno prostornino), vsi poligoni, ki sestavljajo polieder, pa so planarni in preprosti (tj. ne vsebujejo lukenj). Poliedri so tipi poligonskih mrež, ki se najhitreje procesirajo in upodabljajo, zato je večina grafičnih sistemov omejena prav nanje.

Za te tipe mrež velja, da ne obstaja rob, ki bi si ga delili več kot dve stranici, oziroma velja še več, število vseh točk in stranic skupaj je natančno dvakrat večje kot število robov, kar lahko simbolično zapišemo z Eulerjevo enačbo:

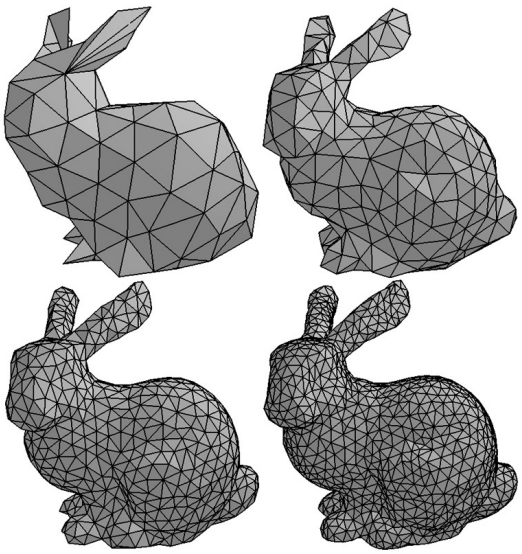
$$V - E + F = 2$$

Simboli v enačbi predstavljajo naslednje: **V** predstavlja število vseh točk, **E** število vseh robov in **F** število vseh stranic poliedra. Za primer vzemimo najpreprostejšo obliko poliedra: tetraeder, ali preprosteje, enakostranična

piramida. Za enakostranično piramido velja, da ima štiri oglišča, štiri stranice in šest robov; 4 - 6 + 4 = 2 [7].

2.3 Poligoni

Kot že prej omenjeno, poligonsko mrežo sestavlja množica točk v tridimenzionalnem prostoru, kjer se več parov točk združuje v daljice ali robove, nadalje skupek robov tvori stranico ali poligon. Torej skupek treh ali več točk v prostoru sestavlja poligon. Večje število poligonov sestavlja poligonsko mrežo, skupek več poligonskih mrež pa tvori končno obliko poljubnega 3D predmeta. Poligoni so običajno kar definirani kot trikotniki, štirikotniki ali drugi preprosti konveksni mnogokotniki, saj njihova oblika olajšuje nadaljnje geometrijske operacije programske ter strojne opreme nad njimi.



Slika 2.1: Število poligonov poljubne poligonske mreže definira njeno natančnost in obliko objekta, ki ga ponazarja

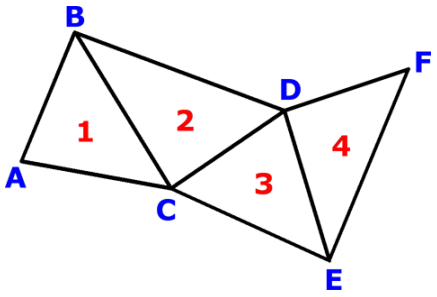
V splošnem so grafični algoritmi za izvajanje geometrijskih operacij omejeni na planarne mnogokotnike, to so taki mnogokotniki, čigar točke oziroma oglišča ležijo znotraj iste ravnine [1]. Trikotnik je sestavljen iz treh točk,

ki definirajo njegova oglišča, je konveksen, kar pomeni, da vsaka potegnjena črta znotraj robov trikotnika v celoti leži znotraj slednjega, zato je edini poligon, ki zagotavlja planarnost. Ravno te lastnosti zagotavljajo trikotniku, da je n-kotnik, ki z minimalno količino informacije zadostuje zapisu ravnine. Posledično iz tega sledi glavni razlog, zakaj so trikotniki primarna izbira za definicijo poligonov arhitektur grafičnih cevovodov. Grafični cevovod je konceptualni model v računalniški grafiki, ki opisuje, katere korake ali faze mora grafični sistem izvesti za pretovrbo grafičnega modela iz podatkovne strukture, ki hrani informacijo o njem, v rastersko ali bitno sliko, ki je kasneje prikazana na zaslonu računalniškega sistema.

Vendar tu naletimo na protislovje, višje smo omenili, da so poligoni poljubne poligonske mreže, lahko tudi poljubni konveksni mnogokotniki, ravnokar pa smo opisali, da so grafični algoritmi, ki izvajajo geometrijske operacije nad njimi, omejeni zgolj na trikotnike. Vsaka poligonska mreža je lahko sestavljena iz poljubnih mnogokotnikov, ki definirajo njene poligone, niti ni nujno, da so konveksni niti planarni. Vendar bo grafična procesna enota v določeni fazi grafičnega cevovoda te mnogokotne poligone modela razbila na trikotnike in nad njimi izvedla željene operacije.

2.4 Trikotniški trakovi

S tem ko smo definirali trikotnike kot primarno obliko definicije poligona, lahko izkoristimo njihovo dodatno lastnost, ki nam zmanjša porabo pomnilnika za reprezentacijo potrebnih podatkovnih struktur za opis geometrije ter topologije poljubne poligonske mreže oziroma predmeta. Govorimo o povezovanju trikotnikov v **trikotniške trakove** (angl. triangle strips), s katerimi opišemo poljubno poligonsko mrežo. Trikotniški trak je definiran kot vrsta zaporedno vezanih trikotnikov, ki si med seboj delijo točke, kar nam omogoča učinkovitejšo uporabo pomnilnika pri shranjevanju podatkov poligonske mreže. Glavni razlog za njihovo uporabo je zmanjšanje količine podatkov, potrebnih za ustvarjanje zaporednih vrst trikotnikov. Število shranjenih točk v pomnilniku se zmanjša s 3N na N + 2, kjer je N število trikotnikov, potrebnih za izris. Posledično temu sledi, da se skrajšajo tudi časi prenosa grafičnih modelov v računalniški pomnilnik (RAM).



Slika 2.2: Poligoni shranjeni v podatkovni strukturi; trikotniški trak

Za preprost primer si oglejmo zaporedje trikotnikov na zgornji sliki 2.2. Brez uporabe trikotniškega traku za predstavitev njihovega zaporedja bi slednje interpretirali ter shranili kot štiri ločene trikotnike: ABC, CBD, CDE in EDF. Vendar z uporabo trikotniškega traku lahko slednje shranimo kot zaporedje točk ABCDEF. Zaporedje točk bi se nato dekodiralo v zaporedje trikotnikov ABC, BCD, CDE in DEF, kjer bi nato vsak sodo indeksiran trikotnik (štetje začenši z 1) obrnil smer svojega zaporedja ogljišč, kjer bi rezultat predstavljal prvotno zaporedje trikotnikov.

2.5 Low poly

Low poly je poligonska mreža v 3D računalniški grafiki, ki je sestavljena iz relativno majhnega števila poligonov. Kot oblikovni stil je namenjen za opisovanje grafičnih objektov z minimalnim obsegom uporabljene geometrije za dosego danega cilja.
Tu se najbrž bralec vpraša, kaj točno pomeni „majhno“ število poligonov. Za poljubno poligonsko mrežo ali model ni točno določenega praga, ki bi kot število točk zaznamoval mejo med low in high poly geometrijo. Število je subjektivne predstave in je relativno odvisno od naslednjih faktorjev [10]:

- Čas, v katerem so bili modeli narejeni, ter za kakšno stopnjo zmogljivosti strojne opreme naprave so bili narejeni (namizni računalnik, konzolna naprava, tablična ali mobilna naprava...).
- Stopnja natančnosti aproksimacije nekega modela.
- Oblika ter lastnosti nekega objekta, iz katerega želimo oblikovati model.

Sam sem mnenja, da če jasno in neločljivo opazimo kote, ki mejijo različne poligone opazovane poligonske mreže, potem je ta najverjetneje narejena v low poly geometriji.

2.5.1 Uporaba v aplikacijah z izvajanjem v realnem času

Uporaba low poly oblikovnega stila je predvsem uporabna v aplikacijah, ki tečejo v realnem času (angl. real time), največkrat v računalniških igrah, kjer je število poligonov, ki jih je grafični sistem zmožen v realnem času obdelati, omejeno s strani strojne opreme grafičnega sistema. Časovna in prostorska zahtevnost ali kompleksnost nekega modela je linearno pogojena s številom poligonov, ki ga sestavljajo. Več poligonov poljuben model šteje, več operacij mora grafični sistem izvesti za dosego izrisa tega na 2D zaslon. Če aplikaciji,

ki teče v realnem času, posredujemo model, ki šteje zelo veliko število poligonov, za katerega grafični sistem ni več zmožen izvajati zahtevanega števila operacij, opazimo efekt zakasnitve prikazovanja željene vsebine na zaslon. Slednje človeško oko ne zaznava več kot gladko gibanje in za tako aplikacijo ne moremo več reči, da na zaslon, v realnem času, dovaja željeno vsebino. Človeško oko zaznava premike v multimedijskih vsebinah, ki se izrisujejo na zaslon, kot gladko gibanje nekje pri 24 sličicah na sekundo (angl. frames per second), kar je tudi eden osnovnih standardov filmske industrije. Ponavadi kot gladko gibanje v računalniških igrah in podobnih aplikacijah, ki tečejo v realnem času, smatramo od 30 sličic na sekundo dalje. To pa zato, ker morajo aplikacije sprejemati vhodne podatke, jih obdelati ter v zadovoljivem času tudi prikazati na zaslonu. Z drugimi besedami povedano, predpostavimo, da takšna aplikacija teče pri 24 sličicah na sekundo, kar pomeni, da se zdi vsaka akcija, ki jo uporabnik izvede, počasna in neodzivna, saj naši možgani pričakujejo dosti hitrejši (realen) odziv, kot pa ga je aplikacija zmožna dovajati. Multimedijske vsebine filmske industrije ne potrebujejo uporabniške interakcije in tako nikakršnih vhodnih podatkov, zato naši možgani gibanje pri 24 sličicah na sekundo prepoznavajo kot konstantno in gladko.

Slika 2.3: Igra Floatlands razvita s strani slovenske ekipe, ki uporablja low poly stil za predstavitev njene vsebine

2.5.2 Zgodovinski presek uporabe

Včasih, ko je bila zmogljivost strojne opreme računalniških sistemov dosti nižja, kot jo poznamo danes, je bil glavni razlog za uporabo tega stila, doseganje sprejemljivo gladkega prikazovanja vsebine na zaslon. Proizvajalci računalniških iger so tako morali zmanjšati število poligonov na sprejemljivo velikost za poganjanje na takratnih sistemih in pri tem ohraniti informacijo vsebine, ki so jo želeli prenesti končnemu uporabniku. Omeniti je potrebno, da je evolucija grafike oziroma napredek v strojni in programski opremi računalniških sistemov močno pripomogel k temu, da je bilo to, kar danes sprejemamo kot low poly stil, včasih vsekakor sprejeto kot napredna grafična upodobitev vsebine z visokim številom uporabljenih poligonov ali high poly.

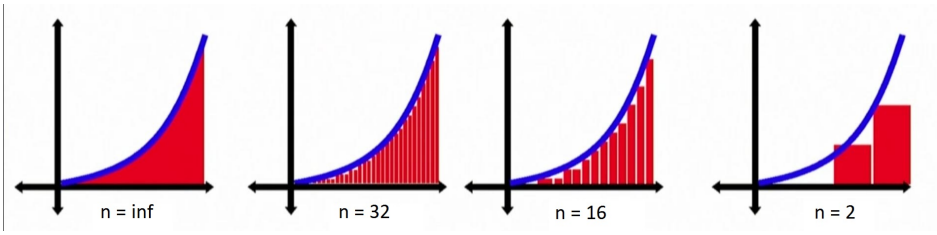
Danes stil še vedno uporabljamo za doseganje sprejemljivo gladkega prikazovanja vsebine v multimedijskih aplikacijah, ki tečejo v realnem času, vendar pa je danes razlogov za uporabo slednjega več. Da se vedno več razvijalcev digitalnih iger odloča za uporabo tega grafičnega stila je v večini primerov povezano z zamanjšanjem stroškov razvoja projektov.
Z uporabo low poly stila smo lahko dosti produktivnejši, v smilsu, da dosežemo hitrejši prenos željene informacije v večjo količino narejene vsebine, kot bi jo dosegli v high poly geomteriji. Slednje je zelo pomembno za podjetja ali skupnosti, ki štejejo manjše število zapolsenih razvijalcev, še toliko pomembneje pa za neodvisne razvijalce iger (angl. independent developer), kateri so popolnoma odvisni zgolj sami od sebe.
Velja omeniti tudi dejstvo, da z njegovo uporabo lahko varčujemo na nivoju računalniških resursov, ki jih ima vsaka naprava oziroma sistem omejeno končno in lahko le te uporabimo na drugih področjih našega projekta za komputacije poljubne vrste podatkov ali pa lahko na ta način razširimo svoj produkt na širši trg (mobilne naprave). Dober primer tega je varčevanje z resursi pri projektih, ki uporabljajo tehnologijo navidezne resničnosti (angl. virtual reality ali VR). Potreba po računalniških resursih takih projektov je običajno podvojena, saj mora „virtual headset“ oziroma naprava, ki jo ima

uporabnik na glavi, za vsako oko prikazovati vsebino pri višjemu številu sličic na sekundo (zaradi potreb gladkih prehodov pri premikih glave) in to na dveh ločenih zaslonih. Ena izmed glavnih prednosti uporabe stila low poly ter drugih alternativ realističnemu upodabljanju vizualne vsebine je tudi ta, da se stil ne „stara“, zgolj zaradi dejstva, ker se ne opira na nenehno realistično aproksimacijo modelov, za katere se tehnoogija iz leto v leto izboljšuje ter s tem tudi viša prag upodabljanja realizma.

2.5.3 Smernice grafičnega stila

Ko oblikujemo modele željene vsebine moramo paziti na smernice pri oblikovanju low poly modelov, ki nas vodijo do kvalitetnejših končnih rezultatov, zato si v nadaljevanju oglejmo nekaj takšnih pristopov.
Sprva naj omenimo, da low poly stil ni kompatibilen kar s katerim koli objektom, sceno ali katero koli drugačno obliko vizualne predstavitve, ki si jo je oblikovalec zamislil za oblikovanje v opisanem stilu. Priporočeno je, da se zavedamo tega „pravila“, preden se lotimo samega oblikovanja. Nekaj primerov, v katerih se stil lepo in dobro izraža ali je stilsko kompatibilen z, so: origami, kubizem, mesta, narava, fantazijski svetovi, itd...
Pri oblikovanju v stilu low poly se moramo zavedati, da morajo naši modeli oziroma scena ponazarjati abstrakcijo nekega pomena, s katerim ne želimo natančne, realne predstavitve modela v svetu. Velja kot nekakšen „digitalni impresionizem“. Kot so nekoč impersionistični slikarji ustvarjali vsebino, ki je bila v nasprotju s pravili in vrednotami klasičnih realističnih slik tistega obdobja, je tudi low poly stil v kontrastu z močno „hiper“ realistično grafiko, ki je v današnjem času pričakovana in upodobljena v modernih računalniških igrah. Razvijalci igričarskih vsebin z vedno bolj realistično grafiko stremijo bolj in bolj k realizmu ter se opirajo na meje zmožnosti strojne opreme, danes vse hitreje razvijajoče se tehnologije. Z low poly stilom pa želimo, nasprotno, uporabniku ponuditi preprostost namesto nasičenosti scene z velikim številom objektov in ga prepustiti njegovi domišliji za zapolnitev vrzeli določene prikazane grafične vsebine.

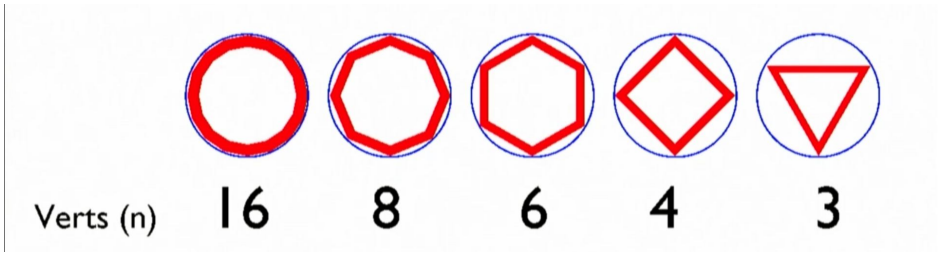
Priporočen potek dela oziroma oblikovanja poljubne vsebine je tak, da imamo že v začetnih fazah oblikovannja v mislih ali pa skicirano na listu papirja, neko grobo predstavitev ideje, kjer iz nje črpamo glavne točke zanimanja, ki predstavljajo njen pomen in jih upodabljamo v razvijajočem se modelu. Nadalje čez model iteriramo, v smilsu, da mu dodajamo, popravljamo, odstranjujemo že obstoječe poligone ter s tem dodajamo in izboljšujemo vsebino objekta, dokler ne pridemo do končne stopnje predstavitve ustvarjenega dela, ki predstavlja končni pomen. V vsaki iteraciji moramo paziti, da ne kršimo prej omenjenih smernic stila in pri tem ne uporabimo prevelikega števila poligonov za ponazoritev naše ideje.
Pri oblikovanju željene vsebine se moramo zavedati njene ločljivosti ali z drugo besedo stopnje njene aproksimacije. Pomen tega je lepo prikazan na sliki 2.4, kjer si lahko predstavljamo željeno vsebino kot rdečo površino pod modro krivuljo, ki je predstavljena z zaporednjem stolpcev. Z neskončnim



Slika 2.4: Predstavitev pojma ločljivosti podatkov

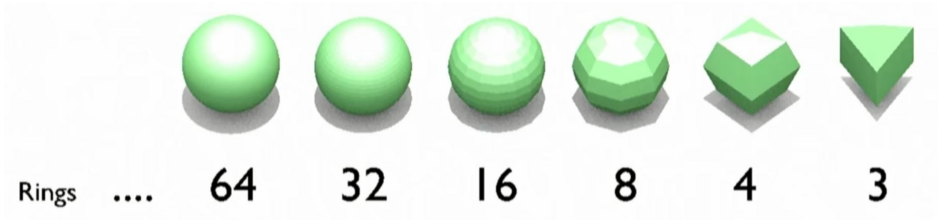
zaporedjem stolpcev lahko prikažemo vsebino brez izgube informacije, če za predstavitev vsebine uporabimo končno število stolpcev, natančneje 32 teh, pri njeni predstavitvi že izgubimo nekaj informacije, vendar zanemarljivo malo. Manj stolpcev kot vzamemo za predstavitev vsebine, več informacije bomo izgubili in manj podatkov bomo potrebovali za njeno predstavitev. Zadnji graf prikazuje kako smo na račun prevelikega prihranka podatkov izgubili preveč informacije, iz česar sledi, da je vsebina neločljiva. Razširimo slednji koncept na obliko lika ter posledično na stil low poly, kar je prikazano na sliki 2.5. Če za predstavitev okroglega lika oziroma kroga uporabimo

16 točk, jasno opazimo, kaj želimo predstaviti. Tega ne moremo več reči, če za predstavitev kroga uporabimo štirikotnik ali celo trikotnik. Pri uporabi šestkotnika lahko opazovalca že prepričamo, da želimo z njim upodobiti krog, seveda gledano v kontekstu, v katerem predstavljamo željeno vsebino. Prejšnje bo veljalo, če vsebino predstavljamo z low poly geometrijo, kjer



Slika 2.5: Predstavitev pojma ločljivosti lika

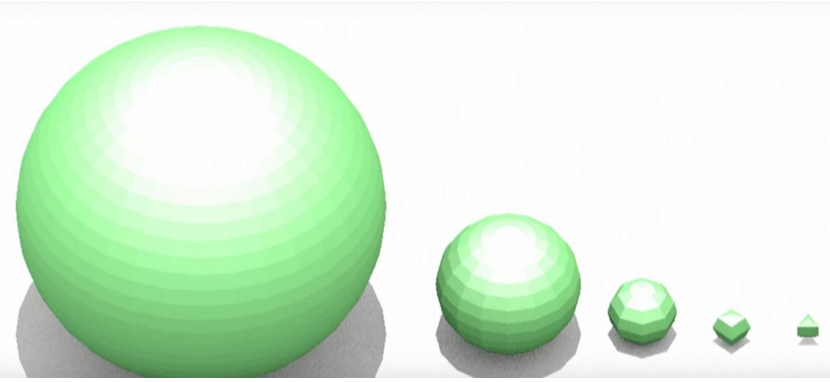
opazovalec ve, da gleda abstraktne ter približne predstavitve objektov in ne točno take, ki obstajajo v realnem svetu. Opaženo ni res, če vsebino predstavljamo, v kontekstu v katerem prevladuje high poly geometrija.
Nadgradimo prejšnji koncept v tridimenzionalen prostor, kjer opazimo, da model z osmimi obroči še zadostuje pojmu krogle, kot je prikazano na sliki 2.6. Strinjamo se lahko, da prejšnje že več ni res za objekta s štirimi in tremi obroči.



Slika 2.6: Predstavitev pojma ločljivosti objekta

Ponovno razširimo ugotovljeno na zaporedje objektov na sliki 2.7. Objekti na tej sliki so popolnoma enaki kot prej, razlika je zgolj v njihovi velikosti ter velikosti njihovih poligonov. Najmanjša velikost poligona v sceni je

enaka najmanjši velikosti poligona vsakega objekta ali povedano drugače, vsi objekti si delijo enako velikost najmanjše velikosti poligonov, iz katere so sestavljeni. V praksi ponavadi za opisani princip velja, da je velikost poligonov nekega obsega poligonske mreže ali scene približno enako velika.

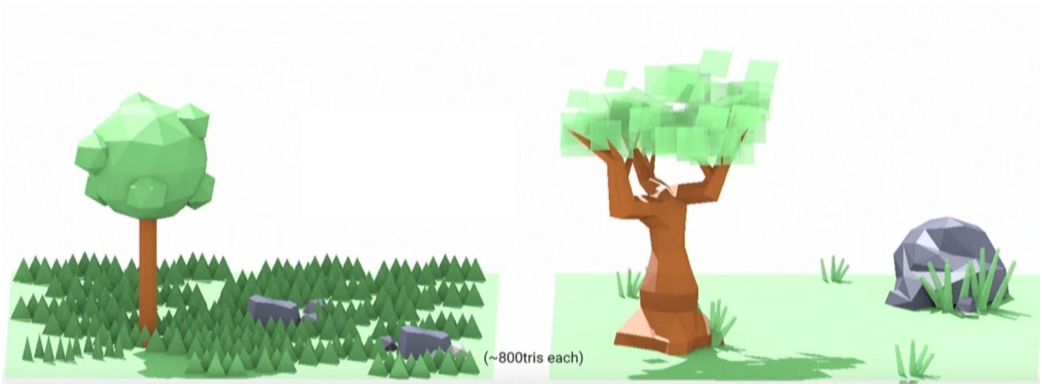


Slika 2.7: Predstavitev pojma atomne velikosti objektov

Sedaj pa lahko opazimo, da objekta s štirimi in tremi obroči ponazarjata bolj okroglo upodobitev krogle, kot sta jo ponazarjala v prejšnjem primeru. Scena namreč nudi sedaj opazovalcu povezanost, konsistenčnost, kontekst, v katerem objekti obstajajo, na katerega se opazovalec lahko sklicuje pri razbiranju, kaj naj kakšen objekt predstavlja. Prejšnje lahko poimenujemo kot (globalna) atomična velikost objektov. Lokalna atomična velikost objektov pa velja, kadar upoštevamo atomično velikost objektov za lokalno omejene objekte.

Poglejmo si še zadnji primer, kako lahko z enakim številom točk v sceni, z drugačno kompozicijsko postavitvijo le teh, ter z različnimi pristopi uporabe zgoraj navedenih smernic popolnoma drugače prikažemo opazovano vsebino. Na sliki 2.8 opazimo dve podobni sceni s popolnoma različnimi lastnostmi ponazoritve vizualne informacije, ki s tem sporočata medsebojne razlike. Obe sceni vsebujeta enako število točk za predstavitev vsebine, in sicer obe natančno osemsto trikotniških poligonov. Takoj opazimo, da imata sceni zelo drugačno celostno podobo objektov v smislu podrobnosti, ki opisujejo vsak

objekt v sceni. Vizualna informacija prve je močno razpršena po celotni sceni, kjer je vsak objekt predstavljen z majhnim obsegom podrobnosti, ki ga opisujejo. Nasprotno, v drugi sceni takoj opazimo, kako je skoraj celotno število točk scene uporabljeno za predstavitev zgolj dveh glavnih predmetov zanimanja v sceni, in sicer drevesa ter skale v ozadju, ki je obdana z nekaj listi zelenja. Slednje je doseženo z uporabo zgoraj navedenega principa atomičnih velikosti objektov v sceni. Prva scena uprizarja atomično velikost objektov na globalni ravni, saj opazimo, da so velikosti poligonov primerljivo enako velike skozi celotno sceno, drugače opazimo v desni sceni, kjer je atomična velikost objektov lokalno omejena na različne predmete v sceni, in sicer: skalo v ozadju, zelenje, krošnjo ter deblo drevesa. Prva scena ponuja veliko količino informacije razpšene skozi veliko število zelo preprostih objektov, katerih podrobnosti so minimalne. Ta tehnika je uporabna, kadar želimo ponazoriti področja visoke gostote vizualne informacije, kjer se pri tem ne spuščamo v podrobnosti posameznih modelov. Tehnika je idealna za predstavitev raznoraznih mest, gozdov, narave v splošnem, povsod kjer si želimo velikega števila objektov z majhnim obsegom podrobnosti. Druga kompozicija ponuja svežino in jasnost, kar je zelo uporabno pri načrtovanju stopenj (angl. level design) v računalniških igrah, saj lahko zaradi zmožnosti očesa, ki samodejno naravno najde predmete zanimanja v sceni, sporočamo igralcu, kam naj gre v danem trenutku, za nadaljevanje poti v stopnji oziroma ga vodimo do željenih točk zanimanja [8].



Slika 2.8: Primerjava med scenami low poly

Poglavje 3

Proceduralno generiranje vsebine

3.1 Definicija

Pojem proceduralna generacija vsebine je definiranan kot ustvarjanje računalniške vsebine z uporabo algoritmov z omejeno ali posredno prisotnostjo uporabnikovega vnosa. Povedano drugače, proceduralno generiranje vsebine se nanaša na uporabo računalniške programske opreme za ustvarjanje njegove vsebine popolnoma samostojno ali v prisotnosti enega ali več razvijalcev. Od tod dalje bomo vedno, ko se bomo sklicevali na opisani pojem, zanj uporabljali kratico PCG (angl. procedural content generation). Izraza „proceduralno“ in „generiranje“ se navezujeta na dejstvo, da imamo pri ustvarjanju vsebine opravka s procedurami ali algoritmi. Koncept PCG poganja računalniški sistem na podlagi trdno postavljenih pravil ter parametrov in izda željeno vsebino. Sistem PCG je tak sistem, ki vključuje metodo PCG kot enega izmed njegovih glavnih delov, kot vir kreiranja vsebine [22].

Vir [22] kot PCG navaja:

- Programsko orodje, ki ustvarja ječe za akcijsko igro, kot je The Legend of Zelda, brez kakršnega koli človeškega vnosa. Vsakič ko se orodje

zažene, ustvari nov nivo igre.

- Sistem, ki ustvarja nova orožja v vesoljski igri kot odgovor na to, kaj igralci počno v igri, tako, da orožja, ki jih sistem generira so evolucijske verzije orožij, ki jih igralci vidijo kot zanimiva in zabavna za uporabo.
- Program, ki ustvarja celotne, igralne in primerno uravnotežene namizne igre popolnoma samostojno, mogoče zgolj uporablja za začetne točke ustvarjanja pravila že obstoječih namiznih iger.
- Igralni pogon, ki za vsebino igre naseljuje njene površine z vegetacijo.

Sedaj pa si oglejmo, kaj vir [22] opredeljuje kot vsebino, ki ne spada pod PCG:

- Zemljevid v strateški igri, ki omogoča uporabniku postavljanje in odstranjevanje predmetov na/iz njega, brez kakršnega koli njegovega odločanja ali samostojnega ustvarjanja.
- Nasprotnik z umetno inteligenco v namizni igri.
- Igralni pogon, zmožen integracije avtomatskega generiranja vegetacije.

3.2 Prednosti in slabosti

Če želimo poljubno računalniško igro dobro podkrepiti z veliko različnimi stopnjami oziroma gledano širše, jo podkrepiti s čim različejšo vsebino, potem moramo izdelati veliko različnih vrst vsebine. S tem igralcu zagotavljamo „svežino“ na vsakem koraku igre, saj igralca ne želimo znova in znova zasititi z enako igralno izkušnjo ter pri tem spremeniti nekaj malenkostnih podrobnosti, da bi ga prepričali, da je svet, v katerem se trenutno nahaja, dovolj različen od prejšnjega. S tem v mislih preidimo na naslednjo prednost uporabe PCG ter dodajmo naboru prednosti uporabe PCG še princip diverzitete ali naključnosti. Če v sistem PCG dodamo element naključnosti kot enega izmed vhodnih parametrov ali pogojev pri kreiranju vsebine, s tem znatno

pripomoremo k ustvarjanju venomer nove, sveže vsebine.
Uporaba koncepta PCG za generiranje vsebine ima tudi to prednost, da končni produkt zaseda malo prostora na trdem disku računalniškega sistema, ravno zaradi majhnih velikosti podatkov projekta, saj PCG preko nabora vhodnih podatkov vsakič znova sam generira vsebino na osnovi podanih pravil. Posledično se s tem tudi zmanjšajo časi nalaganja vsebine na računalniški pomnilnik, iz katerega nadalje PCG jemlje vhodne podatke in izdaja vsebino.

Slabost PCG pristopa je, da zahteva zelo dobro razumevanje problema, za katerega želimo ustvariti vsebino. V kolikor koncepta ne razumemo dovolj dobro, potem sistema PCG ne moremo podkrepiti z vsemi konkretnimi informacijami, pravili in pogoji in pričakovati, da bo le ta ob opiranju na predpisana pravila na izhod uspešno izstavljal načrtovano vsebino.
Prav tako lahko kot slabost opredelimo nezmožnost zavesti, instinkta in „občutka“ danega sistema. S tem želimo sporočiti, da sistem, ki se zgolj opira na pravila in pogoje, ki mu jih je za izdajanje vsebine predpisal človek, nima vseh potrebnih informacij in tako ni zmožen izdajanja enakovredno kvalitetnega odločanja in posledično dela, kot bi ga izdal človek, ki se pri ustvarjanju vsebine opira tako na pravila, kot tudi na svoj instinkt ali občutek, ki ga je pridobil na svojem profesionalnem področju iz dolgotrajnega učenja in preteklih izkušenj. Opozoriti je treba, da je takemu sistemu, kot je opisan zgoraj, človek sam predpisal, na katera pravila in pogoje naj se ta opira ob kreiranju vsebine. Umetna inteligenca računalniških sistemov je sedaj bolj kot še nikoli v izjemnem porastu zanimanja ter raziskav in je mogoče, vsaj teoretično, da bi bil sistem zmožen enakovrednega odločanja kot človek, vendar bi ta potreboval veliko časa in zelo veliko vhodnih podatkov ter iteracij za učenje samega sebe, da bi dosegel nivo razumevanja in odločanja človeka, pri tem pa izdajal enakovredno kvalitetno vsebino. Dejstvo je, da bi bil tak postopek učenja PCG sistema drastično dražji in počasnejši, kot pa najem človeka za ustvarjanje vsebine brez pristopa PCG. Slednje pa ne pomeni, da je PCG torej neuporaben in nezmožen dostavljanja primerljivo kvalitetne vsebine kot

pa človek sam. Slednje zgolj pomeni, da v popolnoma samostojni meri, ni zmožen tega početja, združen s človeško pomočjo, ki v naprej pripravi recimo nekaj grafičnih modelov, pa zna ustvariti, s pomočjo nabora algoritmov, zelo čudovit in raznolik svet.

3.3 Pristopi uporabe v igrah

V času, ko so v uporabo prihajali računalniški sistemi, zmožni grafičnega prikazovanja vsebine na računalniški zaslon, so bili le ti močno omejeni z računalniškim pomnilnikom in je bila potreba po PCG praktično nujna, če so razvijalci želeli sploh narediti nekaj, kar bi bilo igralcu za interakcijo vizualno zadovoljivo. Velikosti pomnilnikov ter trdih diskov računalnikov preprosto niso zagotavljali dovolj prostora za shranjevanje velikega števila v naprej pripravljenih stopenj iger in grafičnih modelov, zato so morali biti narejeni algoritmično, saj algoritem zavzame le delček pomnilnika v primerjavi s celotno grafično vsebino, ki se odraža v neki stopnji igre [15].
Igra Rogue iz leta 1980 je bila ena prvih primerov uporabe PCG za generianje vrat, hodnikov, pošasti ter drugih objektov znotraj ječe (angl. dungeon), s katerimi je igralec interagiral.
Igra Spore iz leta 2008, založnika Electronic Arts, je igra simulacije življenja. Igralec se tekom igranja igre vseskozi sprehaja skozi faze življenja (celična, stvaritvena, civilizacijska...). V stvaritveni fazi mora sestaviti bitje iz različnih delov telesa. Pri tej interakciji so razvijalci uporabljali PCG tehnike za omogočanje kompatibilnosti sestave večjega števila udov skupaj. Natančneje, uporabljali so „interpretacijski“ tip PCG, kjer so čez obstoječe končno število točk, ki definirajo ogrodje bitja (angl. skeleton), izvedli algoritem, ta pa jim je vrnil kompleksnejše ogrodje z dodano vsebino [9].
Zelo zanimiva in napredna uporaba PCG je krojila usodo ene novejših iger, ki je predvsem v svetu video iger sprožila zelo veliko zanimanja ob njenem prihodu, igra pod imenom No Man's Sky. Glavna zanimivost igre je uporaba PCG za ustvarjanje skoraj celotne vsebine igre. Uspeli so proceduralno

ustvariti celotne galaksije, sestavljene iz zvezd, planetov, kateri vsebujejo unikatne ekosisteme z mnogo različnimi podnebnimi tipi, vegetacijo in poselitvijo z neznanimi vesoljskimi bitji. Zanimivo je dejstvo, da je igra narejena v večigralskem načinu in igralci se lahko srečujejo in vidijo, kaj so drugi igralci že odkrili. Zanimivo je tudi, da je verjetnost srečanja zelo majhna, saj igra podpira $18 * 10^{19}$ planetov. Razvijalci so za proceduralno generiranje vsebine uporabili deterministično algoritem, za katerega velja, da bo za dani vhod vedno izstavil enak izhod. Tako je algoritem z uporabo naključnega generatorja števil velikosti 64 bitov izdal neko naključno število te velikosti kot seme za vhod determinističnega algoritma, kateri je nato proizvedel željeno vsebino in jo prikazal uporabniku. S tem razlogom je tudi zelo malo podatkov shranjenih na strežnikih, kjer igra teče, saj se vsakič, ko se igralec približa že nekemu odkritemu sistemu ali planetu, 64 bitno število, ki predstavlja seme vsebine, prenese iz strežnikov in postavi na vhod determinističnemu algoritmu, ki vsakič znova za to seme generira enako vsebino [11].

Slika 3.1: Igra No Man's Sky podjetja Hello Games

Poglavje 4

Predstavitev uporabljenih tehnologij

V tem poglavju si bomo ogledali uporabljene tehnologije, ki so nam omogočile razviti program za generiranje mesta po prinicipu opisanega postopka PCG ter stila low poly, kateri zaznamuje njegovo grafično podobo. Poglavje bo razdeljeno na dve podpoglavja, v katerih se bomo spoznali z orodjema Unity in Blender. Pri opisovanju Blenderja bomo opisali tudi postopke oblikovanja, ki smo jih uporabili za ustvarjanje grafične vsebine. Pri opisovanju okolja Unity bomo opisali njegove glavne lastnosti in nastavitev našega projekta.

4.1 Blender

Blender [2] je brezplačno in odprtokodno 3D programsko okolje, ki v celoti podpira grafični cevovod in postopke grafičnega modeliranja, animacije, izvedbe simulacij, tehnike sledenja videu in urejanje video podatkov. Vsebuje celo igralni pogon za ustvarjanje preprostejših video iger. Za naprednejšo uporabo je uporabnikom na voljo Blender API, ki je skupek funkcij in procedur za dodatno razširjanje okolja za prilagoditev posebnim potrebam projektov. API omogoča programiranje v okolju Python [16]. Okolje teče na večjemu številu operacijskih sistemov, med njimi so: Windows, Linux in ma-

cOS. Licenca pod skupnim imenom „GNU General Public License (GPL)“ omogoča uporabo okolja Blender popolnoma brezplačno, tako za privatno, kot tudi za komercialno rabo.

4.1.1 Uporabniški vmesnik

Uporabnik se lahko ob prvi uporabi orodja počuti rahlo zmedeno, saj je uporabniški vmesnik obdan z veliko conami, kjer vsaka podpira tako enostavne kot tudi naprednejše funkcije, ki nadalje odpirajo dodatne možnosti danega pogleda. Ob zagonu orodja se uporabniku prikaže začetni osnovni pogled, ki je segmentiran na pet ločenih con, ki so sestavljena iz urejevalnikov, ki so prikazani na sliki 4.1 in so:

1. Urejevalnik informacij hrani uporabne informacije o sceni in druge podatke na katere se lahko uporabnik navezuje pri ustvarjanju vsebine. Prav tako hrani dostop do glavnega menija okolja.

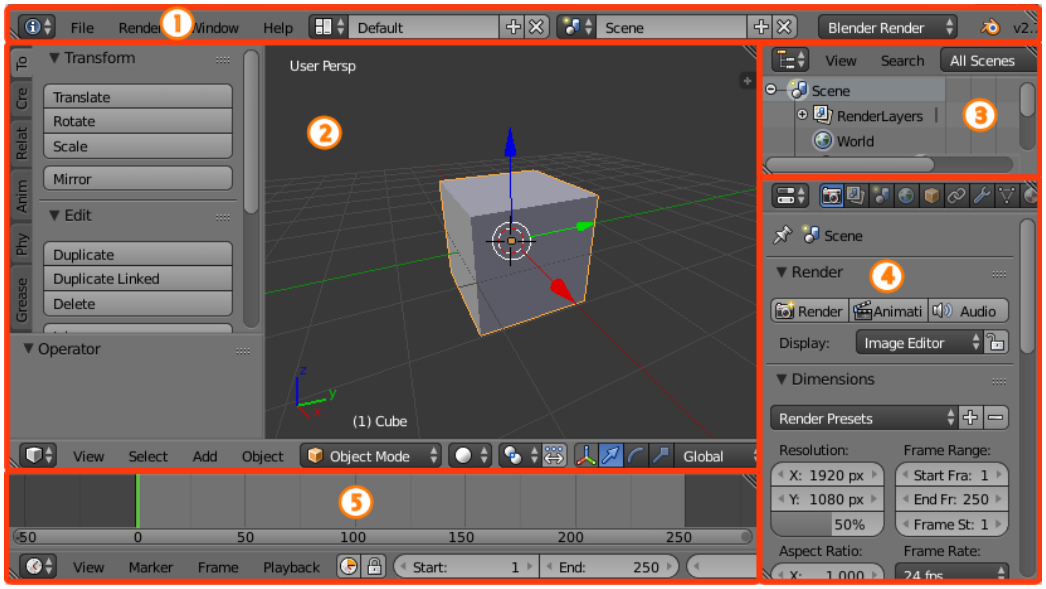
2. Urejevalnik 3D pogleda, ki predstavlja glavno okno oblikovanja vsebine.

3. Urejevalnik „outliner“ je lista, ki strukturno prikazuje podatke o sceni.

4. Urejevalnik časovnega traku je, bolj kot urejevalnik, področje prikazovanja informacije in kontroliranja animacije.

5. Urejevalnik lastnosti, ki nam omogoča urejanje podatkov aktivne scene in objekta.

Hierarhija ali vgnezdenost elementov uporabniškega vmesnika je sledeča: okno, zasloni, cone, urejevalniki, zavihki, plošče, kontrole. Prvotno našteti urejevalniki so prisotni ob osnovnem pogledu, v kolikor se odločimo za drugo zvrst pogleda, so lahko urejevalniki drugačnih oblik. Znotraj urejevalnika informacij najdemo menu, ki se ob kliku razširi in nam poda možnost izbire različnih zvrsti zaslonov. Zasloni so predefinirano urejeni pogledi, ki nam omogočajo različne vrste dela.

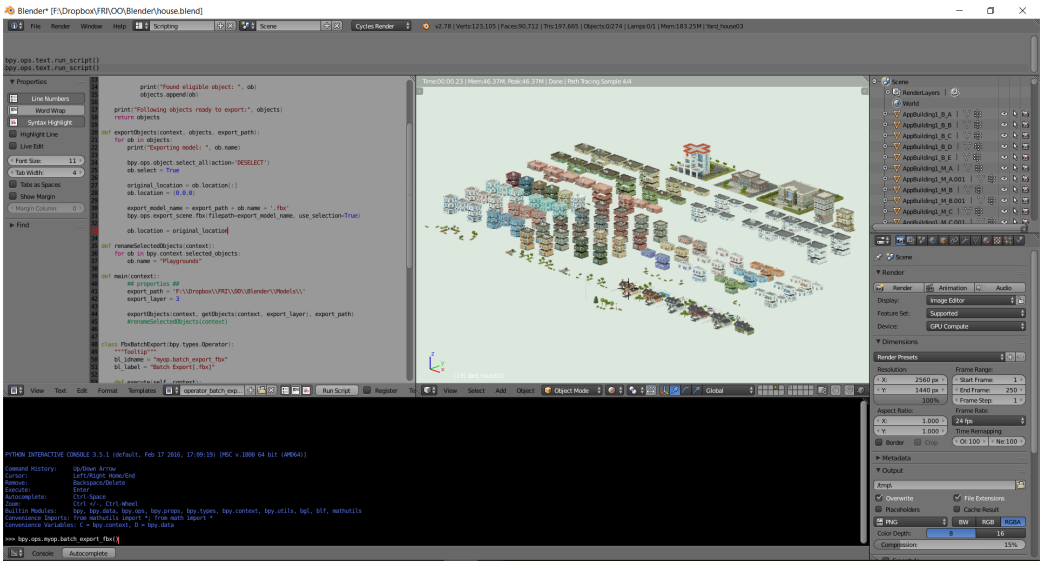


Slika 4.1: Uporabniški vmesnik okolja Blender [3]

Preden nadaljujemo, si oglejmo okna pod imenom „Scripting“, saj smo ga v delu uporabili kot orodje za programiranje lastne skripte, ki nam je omogočila množičen izvoz velikega števila narejenih grafičnih modelov. Njeno natančnejše delovanje si bomo ogledali v nadaljevanju dela. Zaslon nam omogoča dokumentiranje dela ali pa ponuja možnost razširitve osnovnih funkcionalnosti okolja za prilagoditev posameznih potreb projektov. Razširjanje izvajamo preko skript, ki tečejo v programskem jeziku Python. Ob pogledu na okno, prikazano na sliki 4.2, opazimo, da imamo na voljo naslednje urejevalnike:

- urejevalnik 3D pogleda,
- urejevalnik lastnosti,
- sporočilna konzola, ki nam v sosledju prikazuje opise zadnjih izvedenih operatorjev ali akcij,
- urjevalnik teksta, ki nam med drugimi akcijami omogoča tudi pisanje lastnih skript, njihovo poganjanje in tudi shranjevanje,

- konzola python, ki nam omogoča izvajanje ukazov v okolju python, prikazuje zgodovino izvedenih ukazov in možnost samodejnega zaključevanja ukazov.



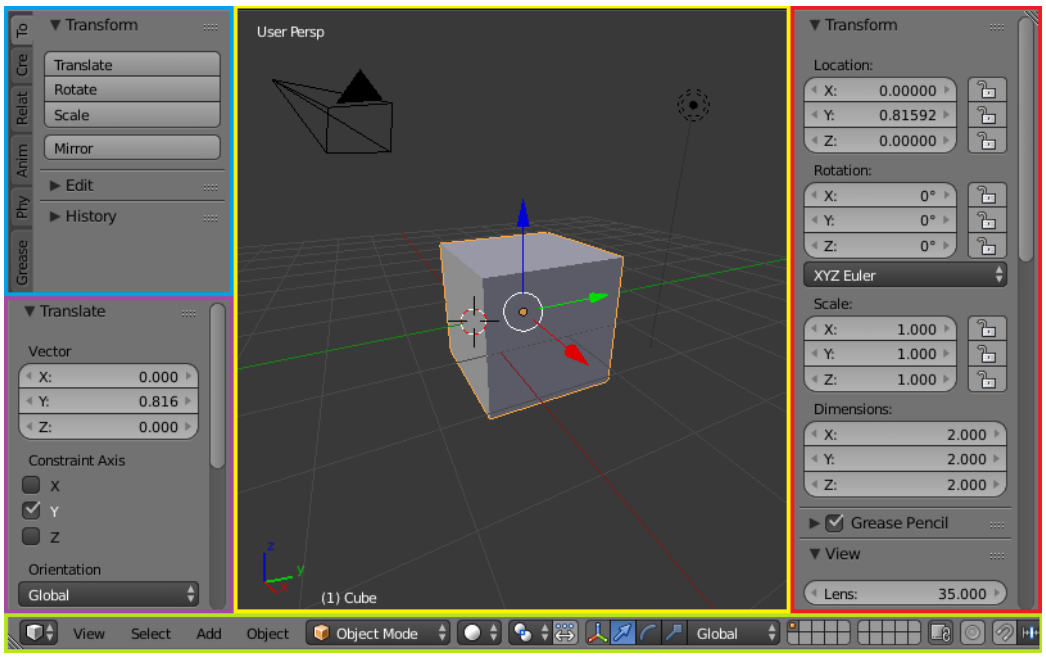
Slika 4.2: Zaslon, ki uporabniku omogoča pisanje in izvajanje skript

Na tem mestu omenimo tudi možnost razširjanja okolja Blender z zunanjimi skriptami ali razširitvami (angl. Add-ons), ki so jih številni uporabniki napisali in objavili na množičnemu spletu zavoljo razširjanja zmožnosti okolja s specifičnimi funkcijami. Pri delu smo tudi sami uporabili nekaj takšnih razširitev, ki smo jih v nadaljevanju tudi opisali. Za namestitev takšne razširitve uporabnik pod nastavitvami zgolj izbere izvor njene datoteke in s tem omogoči njeno uporabo.

V splošnem vsak urejevalnik omogoča različno zvrst pogleda in dela v okolju Blender, ki je ponazorjena na sliki 4.3. Urejevalniki so v splošnem nadalje razdeljeni na naslednje regije:

- glavna regija pogleda (rumena)
- zaglavje urejevalnika (zeleno)

- operatorska plošča (vijolično)
- plošča z orodjem (modro)
- regija z lastnostmi (rdeče)



Slika 4.3: Regije, ki sestavljajo posamezen urejevalnik okolja

Regije lahko za prikazovanje in manipuliranje informacij vsebine vsebujejo strukturirane plošče ali zavihke. Dodatno lahko omenimo, da so plošče najmanjša organizacijska enota, ki jo lahko skrčimo ali razširimo ter s tem prikrivamo/odkrivamo elemente, ki v njej nastopajo. Pri ustvarjanju grafične vsebine smo predvsem dostopali do dveh urejevalnikov, in sicer urejevalnika 3D pogleda in urejevalnika UV/slikovnega materiala, zato si v nadaljevanju natančneje oglejmo njuno delovanje.

4.1.2 Oblikovanje grafične vsebine

Naj opozorim, da zaradi zelo širokega nabora uporabljenih funkcij, ne moremo opisati vseh, ki smo jih koristili, saj bi bilo delo preobsežno. Na sliki 4.4 so prikazane vse funkcionalnosti, ki nam jih ponuja zaglavje urejevalnika, slednje so:

- „Editor type“ ponuja možnost izbire različnih urejevalnikov.
- „Menu“, skupek zavihkov, ki nam omogočajo navigacijo po 3D prostoru, vsebujejo orodja za dodajanje, izbiro ali zajem različnih vrst objektov in omogočajo manipulacijo nad njimi. Pri izbiri geometrije ima uporabnik na voljo nekaj zelo uporabnih možnosti. Geometrijo lahko izbere preko tako imenovane „Box select“ možnosti izbire, kjer uporabnik z miško specificira polje, znotraj katerega zajame željeno vsebino. Prav tako ima uporabnik možnost izbire „povezane“ geometrije, to je vsa geometrija, ki je nastala iz ene instance primitiva. V načinu objekta ima uporabnik možnost dodajanja različnih vrst objektov, kot so: poligonske mreže, krivulje, NURBS površine, kamere, luči, itd... Poligonske mreže v okolju Blender ponujajo različne osnovne oblike (primitive), te so: ravnina, kocka, krog, UV sfera, ico sfera (polieder sestavljen iz trikotnih poligonov), valj, stožec, torus, mreža in opičja glava (maskota okolja). Vsak izmed omenjenih primitivov ima svoj namen uporabe. Sami smo pri oblikovanju zgradb največkrat začeli s kocko ali ravnino in v načinu urejanja nato manipulirali njeno geometrijo. Za predstavitev različnih vrst vegetacije (krošenj dreves in grmičevja) smo začeli gradnjo s primitivom ico sfera.
- „Orientation“ ponuja smer pogleda v tridimenzionalnem kartezičnem prostoru.
- „Active object“ prikazuje ime izbranega objekta.
- „Modes“ vsebuje seznam možnosti izbire vrste manipulacije ene instance objekta v sceni. Pri oblikovanju vsebine smo dostopali do dveh,

in sicer do načina objekta (angl. Object mode) in načina urejanja (angl. Edit mode). V načinu objekta imamo možnost manipulacije nad objektom na globalni ravni, kar pomeni, da nimamo možnosti posega v njegovo geometrijo (položaj, rotacija, velikost, ipd...). Način je tudi uporaben za grupiranje podobnih objektov. Slednje smo uporabili pri gradnji ograje določene hiše, kjer smo zgolj izdelali en kos ograje, jo multiplicirali, rotirali in postavili na željeno mesto v prostoru, ter nazadnje vse te objekte združili v gručo, ki je predstavljala celoten pas ograje, ki je obdajal dano hišo. V načinu urejanja imamo možnost manipulacije geometrije in topologije izbranega objekta. Nad geometrijo in topologijo objekta lahko vplivamo na tri različne načine. Funkcija imenovana „Mesh Selection Mode“ nam v tem načinu ponuja izbiro med točkami, robovi ali stranicami, nad katerimi izvajamo transformacije poligonske mreže, opisane pod pojmom „Transform Manipulator“. Preko slednje kombinacije funkcij dosežemo večino opravljenega dela v 3D modeliranju.

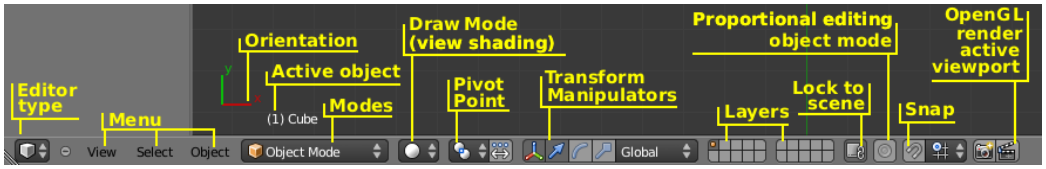
- „Draw mode“ ponuja način izgleda vsebine oziroma način izrisa in senčenja objektov. Največ časa smo uporabljali tako imenovane „Rendered“, „Material“ in „Wireframe“ načine pogleda. Prvi nam izrisuje vsebino z natančnim senčenjem, na katerega vplivajo vse luči v sceni. Izris omogoča še mnogo natančnejših možnosti manipulacije nad senčenjem scene. Drugi način izrisa predstavlja (senči vsebino) s hitro aproksimacijo uporabljenih tekstur. Ta nam je služil kot primarni pogled na vsebino ob njeni manipulaciji v načinu objekta. Zadnji našteti način izrisa, za katerega dostikrat slišimo ime žičnati model, smo v največji meri uporabljali v načinu urejanja. V slednjem smo največkrat potrebovali „transparenten“ pogled na zajem geometrije objekta. Tako smo se postavili v ortografski način in s pomočjo funkcije zajema geometrije zajeli željeno geometrijo objekta.

- „Pivot point“ predstavlja način postavitve pivotne točke ali na kratko

pivota danega objekta. Za postavitev pivota smo uporabljali način mediane, ki ponazarja sredinsko vrednost nad množico podatkov. Ta način nam je pomagal pri identični navpični nastavitvi pivotne točke vsakega nivoja zgradbe.

- „Transform Manipulator“ loči načine transformacije danega objekta ali geometrije (v načinu uerjanja) na: premik, rotacijo in skaliranje.
- „Layers“ so nivoji ali plasti, ki omogočajo prikaz željenih objektov na različnih nivojih v sceni. Pri izdelavi vsebine smo tako zaradi lažjega pregleda nad sceno ločili različne tipe stavb na različne nivoje v sceni, kjer je tako na primer ena vsebovala zgolj rezidenčne zgradbe, druga večstanovanjske objekte, tretja večnadstropne stavbe, itd...
- „Lock to scene“ omogoča delitev uporabe kamer, luči in aktivnih scen, gledano na celotno sceno v okolju.
- „Proportional Edit“ glede na obseg željene sorazmernosti, omogoča sorazmerno urejanje vsebine. V naravi velikokrat opažamo, da okrogle stvari niso popolnoma simetrično in proporcionalno okrogle. Slednja funkcionalnost nam je omogočila ravno to. Sorazmerno urejanje smo tako uporabili pri manipulaciji poligonskih mrež vegetacij okroglih oblik (grmičevja in krošnje dreves).
- „Snap“, ena izmed zelo uporabnih možnosti orodja, ki nam močno olajša in pohitri urejanje geometrije ali objektov kot celote. Funkcija pri translacijah izbrane geometrije omogoča „lepljenje“ le te na izbrano os v sceni, posamezno točko, rob ali stranico. S tem dosežemo, da je geometrija objekta lepo poravnana.
- „OpenGL Render“ gumb nam omogoči izris vsebine v senčenju načina OpenGL. Uporaben je za hiter predogled vsebine v kompleksnejših scenah.

Poleg omenjenih funkcionalnosti zaglavja urejevalnika 3d pogleda smo si pri oblikovanju pomagali tudi z naslednjimi vgrajenimi funkcijami:



Slika 4.4: Zaglavje urejevalnika pogleda 3D [4].

- „Separate“ ponuja možnost ločitve trenutno izbrane geometrije v načinu urejanja na ločen objekt. Slednje smo uporabili povsod, kjer smo želeli iz danega objekta vzeti kos njegove mreže (vrata, okna, luči, dimnik, itd...) in ga uporabiti pri izgradnji trenutnega objekta.

- „Duplicate“ omogoča kloniranje določenega objekta ali njegove vsebine, brez navezovanja na preostalo poligonsko mrežo (v kolikor želimo kopirati le del nje). Uporabili smo ga predvsem, kjer smo izdelali več različnih instanc ene vrste zgradb. Slednjim smo nato spremenili barvo (položaj geometrije na teksturi) in uvedli dodatne spremembe v geometriji.

- „Loop cut“, prav tako ena izmed pomembnejših in večkrat uporabljenih funkcij. Operacija z vstavljanjem poljubnega števila robov, ki obsegajo željeno mrežo, razdeli stranice, ki sestavljajo obsegano mrežo. Uporabili smo jo na primer pri razdelitvi zgradbe na njene sloje, pri ustvarjanju oknatih površin večnadstropnih zgradb, razdelitvi debla drevesa sprva na horizontalen pas ter nato tega še omejili na poligon oblike šestkotnika, iz katerega smo z uporabo funkcije „extrude“ ustvarili vejo drevesa.

- „Limited dissolve“, funkcionalnost, ki je zmožna odstranjevanja/zapolnjevanja oziroma združevanja poligonov okrog nepotrebnih robov. S tem poskrbimo za zmanjšanje obsega uporabljene geometrije. Njena uporaba je priporočljivo prisotna za tem, ko geometrijo nekega objekta transformiramo v željeno obliko, pri tem pa uporabimo operacijo „Loop

cut“, katera po končanem delu pusti veliko poligonov ločenih.

- „Extrude“ je orodje, ki nam duplicira točke izbrane geometrije in obenem obdrži povezavo z njo. Točke se spremenijo v robove in robovi tvorijo novo nastale stranice. Prav tako eno izmed, če ne celo največkrat uporabljeno orodje 3d modeliranja objektov. Uporabljeno povsod, kjer smo želeli 2D površino spremeniti v 3D obliko.

- „Oscurart Tools“ [12], napredno orodje, predstavljeno kot razširitev okolja Blender in nam omogoča število različnih operacij nad objekti. Pri delu smo uporabili njegovo funkcionalnost, ki nam je za izbrano kartezično os enakomerno razporedila izbrane objekte. Slednje smo uporabili povsod, kjer smo želeli enakomerne porazdelitve objektov v zaporedju (okna zgradbe, drevesa v vrsti, luči na ulici, itd...)

- „Razširitev za izbiro pivotne točke v načinu urejanja“. Slednje smo uporabili, ko smo želeli spremeniti pivotno točko vsakega objekta znotraj urejevalnega načina. Potrebno je omeniti, da kot privzeto Blender omogoča slednje početje zgolj v načinu objekta. Ker naš projekt šteje zelo veliko število objektov, smo si z uporabo te razširitve delo močno olajšali.

- „Decimate modifier“ je orodje, ki zmanjša željeno število uporabljenih točk/stranic za predstavitev poligonske mreže izbranega objekta, pri tem pa sledi minimalni spremembi oblike. Orodje smo uporabili pri zmanjšanju števila poligonov krošenj dreves in grmičevja, pri katerih je minimalno število, ki še ponazarja to obliko, zelo pomembno zaradi performančnih razlogov, saj so to objekti okrogle oblike, ki za njihovo predstavitev potrebujejo večje število poligonov.

Vse do sedaj omenjene funkcije so nam pomagale zgolj pri manipuliranju geometrije vsebine, poglejmo si še, kako smo narejeni vsebini v urejevalniku UV/slikovnega materiala dodali teksturo in jo s tem „obarvali“. Vsakemu

objektu, kateremu smo sprva ustvarili geometrijo, smo za tem dodali tudi teksturo. Tak postopek je seveda učinkovitejši, kot če bi sprva oblikovali vse objekte in jim nato predpisali teksturo, saj si veliko objektov lasti določeno vsebino, kateri bi morali znotraj velikega števila objektov večkrat dodati teksturo.
Za dodajanje barv vsebini imamo na voljo 3 možnosti:

- vsako barvo objekta predstavimo z različnim materialom,
- uporabimo način „Vertex Paint“,
- uporabimo eno teksturo, ki bo predstavljala našo celotno vsebino.

Prva metoda predstavlja grafično procesni enoti na nivoju senčenja zelo veliko nepotrebnega dela, saj mora le ta izvesti toliko obhodov senčenj naše scene, kolikor materialov smo uporabili. Druga metoda je v splošnem zelo popularna, vendar ima dve slabi lastnosti. Prva je ta, da Blender ne omogoča zamenjave barve skupini točk, katerim je bila poljubna barva že dodeljena. Druga pa je, da pri opredeljevanju materiala potrebujemo posebne tako imenovane shaderje za predstavljanje barv točk, katere večina igralnih pogonov privzeto ne vsebuje. Odločimo se za zadnjo metodo, saj ima z uporabo le enega materiala performančne prednosti in je enostavna za uporabo. Taki teksturi, ki za predstavitev barv objektov vsebuje večji nabor slik, pravimo slikovni atlas (angl. texture atlas). Pri nas je ta nabor slik zaradi uporabe enoličnih barv predstavljen kot nabor različnih barv velikosti 2x2 piksla. Celotna tekstura je velikosti 64x64 pikslov.

Slika 4.5: Textura, kot slikovni atlas

Kadar poligoni zajemajo neko dejansko sliko teksture, je potrebno površino teh poligonov čez željeno sliko natančno „razkriti“. Temu procesu pravimo „UV mapping“, saj koordinate točk 3D modela, definirane v kartezični obliki kot $p = (x, y, z)$ projeciramo na 2D teksturo, v kateri so točke definirane kot $p = (u, v)$. V našem primeru zaradi enolične uporabe barv tega procesa ne potrebujemo, zagotoviti moramo zgolj, da nastavimo točke koordinat izbranih poligonov na površino v prostoru teksture, ki predstavlja željeno barvo. Temu procesu pa pravimo „texture mapping“.

4.1.3 Izvoz modelov

Sedaj, ko je željena vsebina ustvarjena, jo moramo še izvoziti v obliko, ki jo igralni pogon Unity zna prebrati. Zaradi velikega števila ustvarjenih modelov, natančneje 342 teh, potrebujemo način, ki nam bo v željenem zapisu avtomatsko izvozil dano število modelov. Tovrstno početje bomo izvedli s pisanjem skripte v zaslonu „Scripting“, katerega urejevalnik smo opisali v uvodnem delu poglavja. Preletimo čez vsebino skripte, ki smo jo spisali.

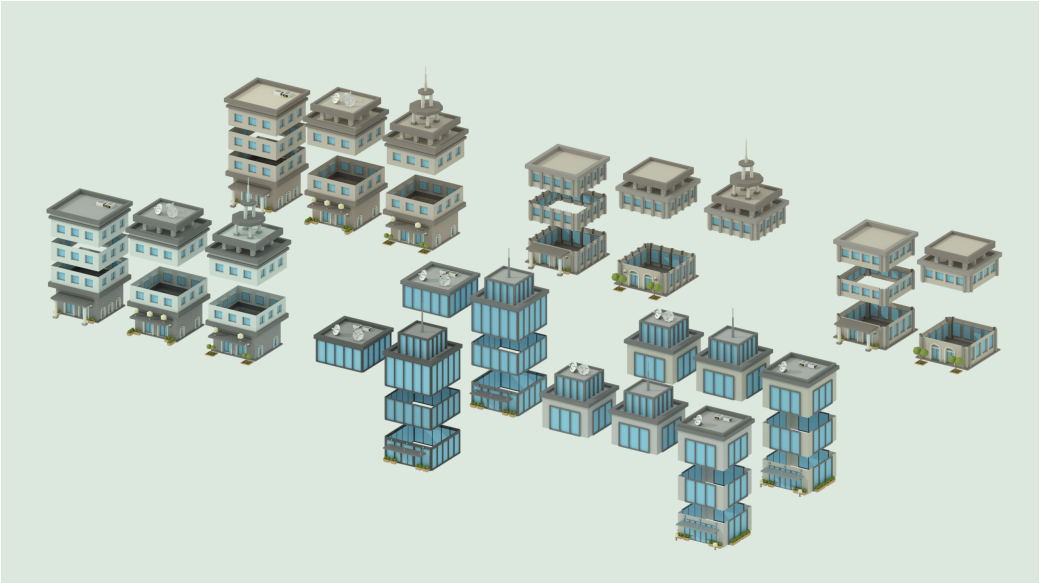
```
import bpy

def getObjects(context, layer):
    objects = []
    for ob in context.scene.objects:
        if ob.layers[layer]:
            if ob.type != 'MESH':
                continue
            if len(ob.data.vertices) == 0:
                continue

            objects.append(ob)
    return objects

def exportObjects(context, objects, export_path):
    for ob in objects:
        bpy.ops.object.select_all(action='DESELECT')
        ob.select = True
        original_location = ob.location[:]
        ob.location = (0,0,0)

        export_model_name = export_path + ob.name + '.fbx'
        bpy.ops.export_scene.fbx(filepath=export_model_name,
        use_selection=True, apply_unit_scale=False)

        ob.location = original_location

def main(context):
        export_path = 'F:\\Diploma\\Blender\\Meshes\\'
        export_layer = 2

        exportObjects(context, getObjects(context, export_layer), export_path)

class FbxBatchExport(bpy.types.Operator):
    """Tooltip"""
    bl_idname = "myop.batch_export_fbx"
```

```
    def execute(self, context):
        main(context)
        return {'FINISHED'}

def register():
    bpy.utils.register_class(FbxBatchExport)

def unregister():
    bpy.utils.unregister_class(FbxBatchExport)

register()
```

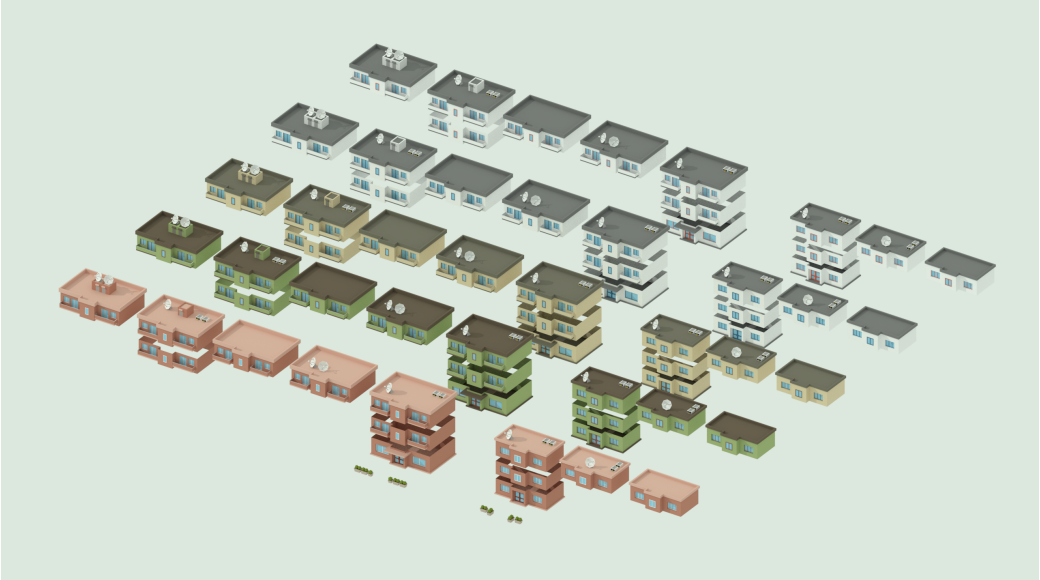
Skripta je napisana v jeziku Python in izkorišča Blenderjev API [5] za pridobivanje in izvajanje vgrajenih funkcionalnosti okolja Blender. Skripta ustvari nov operator, ki je akcija, ki izvaja željeno vsebino. Funkcija register naloži razred „FbxBatchExport“ v okolje Blender in s tem razširi obseg funkcionalnosti okolja. Funkcija „main“ vsebuje dve lastnosti, ki definirata: pot shranjenih modelov ter nivo, iz katerega želimo izvoziti modele. Funkcija „exportObjects“ iterira čez podano listo objektov, izbere trenutni objekt, ga postavi na izhodišče koordinatnega sistema, prebere njegovo ime in ga tako poimenovanega shrani v zapisu .fbx na specificirano pot v računalniku. Ob koncu iteracije se objekt postavi na njegov prvotni položaj. Funkcija „getObjects“ pridobi listo objektov, ki se kot parameter posredujejo prejšnji funkciji. To stori tako, da znotraj podanega nivoja scene iterira čez vse prisotne objekte ter pri vsakem preveri, ali je slednji poligonska mreža in preveri, če ni prazen. V kolikor so opisani pogoji izpolnjeni, objekt doda na listo že najdenih objektov.

4.1.4 Predstavitev modelov

Predstavljena vsebina obsega večje število slik upodabljanj, saj bi ena sama slika, zaradi preobsežnosti scene, izgubila preveč vizualne informacije. Naj še omenimo, da smo vse zgradbe ločili na tri nivoje: spodnji, srednji in zgornji nivo, po potrebi imajo določeni tipi zgradb tudi nivo vegetacije. Tako ureditev smo potrebovali zavoljo potreb PCG, ki smo ga uporabili pri gradnji mesta. Njegovo delovanje je podrobneje opisano v poglavju 5.



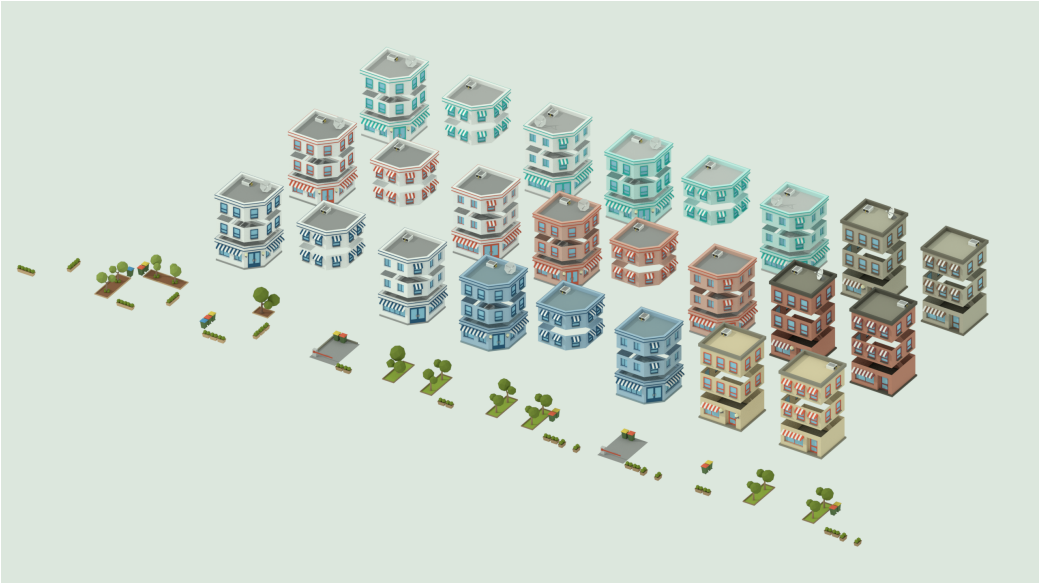
Slika 4.6: Predstavitev večnadstropnih zgradb



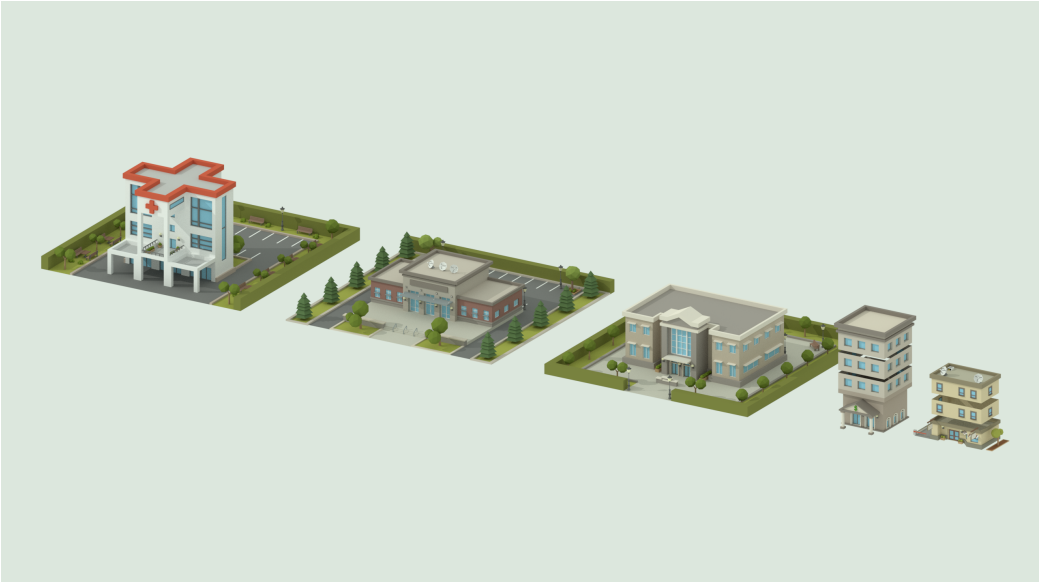
Slika 4.7: Predstavitev večstanovanjskih zgradb

Slika 4.8: Predstavitev poslovnih zgradb

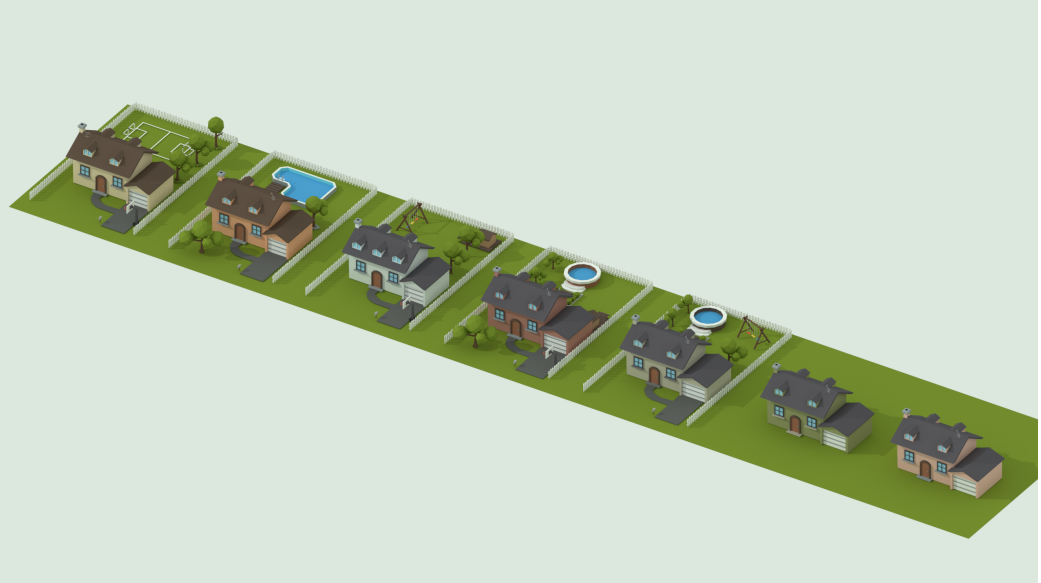
Slika 4.9: Predstavitev nizkih trgovin



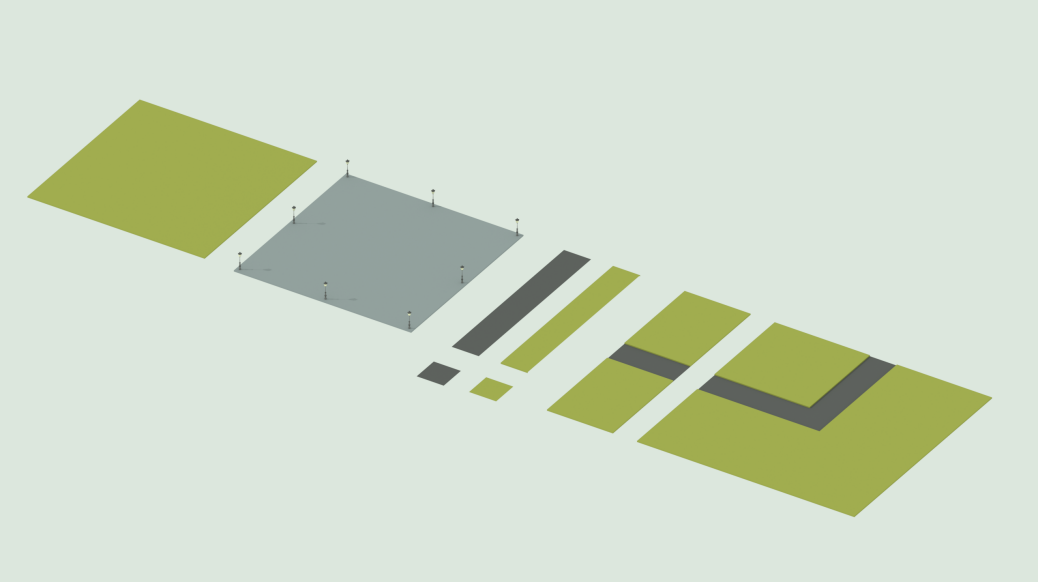
Slika 4.10: Predstavitev višjih trgovin



Slika 4.11: Predstavitev javnih zgradb



Slika 4.12: Predstavitev hiš



Slika 4.13: Predstavitev modelov, na katerih je izvedeno postavljanje objektov, opisano v nadaljevanju

Slika 4.14: Predstavitev ostalih vrst modelov, ki sestavljajo sceno

4.2 Unity

Unity[17] je igralni pogon, ki je primarno osredotočen na razvoj večplatformnih iger. Podpira platforme: Windows, macOS, Linux, Android, iOS, Playstation 4, Xbox One, Oculus Rift... Uporabniku so na voljo štiri licence [20] uporabe:

- Personal, brezplačna in nudi vse funkcionalnosti pogona kot ostale licence z razliko, da nima možnosti uporabe poljubnega začetnega zaslona, ki se uporabniku prikaže ob zagonu dane aplikacije. Je za vsakogar, čigar s prodajo interaktivnih aplikacij, narejenih v pogonu, letni dobiček šteje manj kot dvesto tisoč ameriških dolarjev.

- Plus, je na voljo za petintrideset ameriških dolarjev mesečno. Ponuja razširljivo vgrajeno analitiko, uporabo poljubnega začetnega zaslona ob zagonu, performančnega poročanja. Pogoji uporabe so enaki zgornji.

- Pro, je na voljo za stopetindvajset ameriških dolarjev mesečno. Ponuja vse možnosti pogona kot prejšnja verzija z dodatkom profesionalnega nivoja ponujenih storitev in podpore pri uporabi. Je brez pogojev uporabe.

- Enterprise, je namenjena ekipam, ki štejejo več kot enaindvajset zaposlenih in je namenjena za posebne projektne potrebe. Vse ostale potrebne informacije pridobi uporabnik po dogovoru.

Igralni pogon podpira odlične oblike pomoči pri ustvarjanju vsebine. Uporabniku je na voljo pomoč v obliki video vsebin, dobro opredeljene dokumentacije, letnih srečanj, spletnih izobraževanj, forumov in navsezadnje v trgovini unity. Zadnja je stran pod imenom „Asset Store“, kjer imajo uporabniki možnost nalaganja in prenosa že obstoječe vsebine, ki so jo izdelali drugi uporabniki in jo ponujajo po določeni ceni. Prav tako storitev ponuja „Unity Connect“, ki je poklicna mreža razvijalcev, kjer slednji objavljajo svoje ustvarjeno delo z namenom iskanja novega kadra ali zaposlitve.

4.2.1 Uporabniški vmesnik

Uporabniški vmesnik[18] pogona sestavljajo zavihki, kateri v odvisnosti od njihove zvrsti omogočajo uporabniku prikazovanje in manipulacijo nad vsebovano podatkovno vsebino. Uporabnik ima za prilagajanje potrebam delovnega projekta na voljo veliko svobode pri poljubni razvrstitvi le teh.
Ob prvem zagonu okolja Unity je uporabniški vmesnik sestavljen iz sledečih zavihkov:

- „Project Window“ je projektno okno, ki vsebuje celotno uvoženo vsebino, ki jo ima uporabnik na voljo pri ustvarjanju.

- „Scene View“ okno omogoča pogled in interakcijo nad objekti (grafični modeli, kamere, luči, efekti, in drugi tipi), ki sestavljajo sceno projekta. Pod zavihkom najdemo tudi možnost prikaza tako imenovanih gizmov, ki v sceni ponujajo predogled ikon objektov, prikazovanje žarkov in drugih uporabnih label, ki nam omogočajo lažje navigacije po sceni ali pa so nam v pomoč pri „razhroščevanju“ projekta.

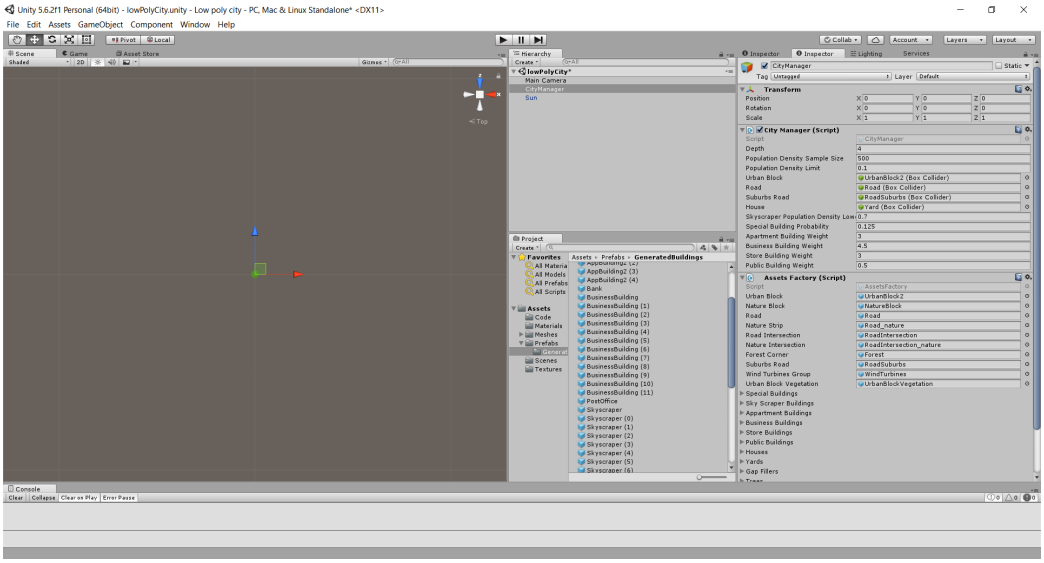
- „Game View“, okno, ki predstavlja končen izgled vsebine igre, senčen skozi leče kamer. Za pogled na željeno vsebino je tako obvezna uporaba vsaj ene kamere v sceni.

- „Hierarchy Window“ predstavlja hierarhičen, tekstovni pregled vseh objektov, ki sestavljajo našo sceno. Prikazuje hierarhično odvisnost objektov (vgnezdenost drug v drugega) med seboj.

- „Inspector Window“ prikazuje in omogoča manipulacijo nad podatki izbranega objekta. Na tem mestu omenimo, da Unity za predstavitev objektov uporablja ime „Game object“, predstavljamo si jih lahko kot zabojnik, ki hrani njegovo željeno vsebino. Ta je predstavljena kot množica uporabljenih komponent (angl. Components), ki opisujejo njeno funkcionalnost. Vsakemu objektu lahko podamo željene tipe

komponent in slednjega povlečemo v projektno okno ter ga tako shranimo kot tako imenovan „prefab“. Tak objekt je predstavljen kot instanca objekta z vsemi dodanimi komponentami in lahko iz njega v realnem času projekta ustvarimo poljubno število novih objektov z isto vsebino. Poljuben objekt je lahko prazen, vendar velja, da vsak objekt vsebuje komponento tipa „Transform“, ki predstavlja njegov položaj, rotacijo in skalo.

- „Toolbar“ zavihek vsebuje orodja manipulacije nad sceno in objekti. Orodja ponujajo navigacijo znotraj scene, transformacijo objektov, nastavitev pivota objekta in nenazadnje tudi funkcije zagona, premora in premika po času naprej v obsegu ene sličice projekta. Je edini zavihek, ki ga uporabnik ne more zamenjati ali premakniti.



Slika 4.15: Uporabniški vmesnik okolja Unity

4.2.2 Nastavitev projekta

Vsa naša vsebina leži znotraj mape imenovane „Assets“, ki se nahaja v projektnem oknu našega projekta. Vsebina je razdeljena po naslednjih mapah:

- **Code**, mapa hrani več razredov programske kode, ki opisje logiko delovanja projekta.

- **Materials**, znotraj nje je material, ki je potreben za uspešno prikazovanje barve ustvarjene grafične vsebine. Slednji je pripet na sleherni objekt, ki je predstavljen s poligonsko mrežo. Material temelji na senčenju spekularnega odboja (angl. specular reflection) z ničelno vrednostjo gladkosti. S tem smo želeli doseči trdo senčenje oziroma tako imenovano tehniko „flat shading“. Tehnika izračuna senčenje vsakega poligona na podlagi kota med njegovo normalo in vpadno svetlobo, njegove zaznamujoče barve in intenzivnosti luči vpadne svetlobe.

- **Meshes**, mapa hrani vse poligonske mreže, ki predstavljajo vsak uvožen model. Naj dodamo, da smo pri uvozu modelov sprva imeli težave pri njihovem prikazovanju, zaradi tehnike izločanja zadnjih ploskev (angl. Backface culling), ki jo grafični cevovod okolja Unity, zaradi performančnih razlogov uporablja pri izrisu vsebine. Tehnika izloči vse ploskve, ki jih kamera ne vidi in pri tem zmanjša obseg vsebine, ki jo mora GPE upodobiti. Težavo smo rešili tako, da smo se v okolju Blender sprehodili čez celotno ustvarjeno vsebino in obrnili normale poligonov, ki so kazale v nasprotno smer (smer normale definira ali naj se željena ploskev izloči ali ne).

- **Prefabs**, mapa hrani vse ustvarjene prefab objekte z različnimi vsebinami.

- **Scenes**, Unity omogoča hranjenje večjega števila scen v projektu, v kolikor želimo tega predstaviti z več nivoji. Mi smo za prikaz mesta potrebovali zgolj eno.

- **Textures**, hrani našo teksturo prikazano na sliki 4.5, zanjo pri mapiranju ne uporabljamo možnosti bilinearnega ali trilinearnega filtriranja. Slednje je pravzaprav vzorčenje tekslov (najmanjša enota v prostoru

teksture) pri mapiranju teksture na nek model, zavoljo gladkih prehodov med poligoni, ki mapirajo različne dele teksture. Zaradi uporabe enoličnih barv tega ne potrebujemo.

Hierarhija scene našega projekta pred zagonom vsebuje tri objekte: kamero, usmerjeno luč in objekt, ki drži vse interaktivne parametre logike PCG, kateri bo natančneje opisan v naslednjem poglavju. Za uspešno prikazovanje grafične vsebine v stilu low poly potrebujemo temu primerno dobro nastavljeno osvetlitev scene. Glavni vir svetlobe v sceni je omenjena usmerjena luč, ki smo jo poimenovali kar sonce, saj dobro imitira njegove lastnosti. Za njegov način oddajanja svetlobe uporabimo način „realtime“, ki izračuna svetlobo v sceni za vsako prikazano sličico. Prav tako nastavimo prikazovanje popolnoma trdih oblik senc. Če opisano luč v sceni ugasnemo, opazimo, da manjši del vira svetlobe prihaja od drugod. Slednjemu pravimo ambientna svetloba in pripomore k globalnemu osvetljevanju scene, za prikaz lepših rezultatov, kamor dejanska svetloba sonca ne seže. Nekatere najpomembnejše lastnosti kamere so: njen zorni kot (opisuje širino kota, skozi katerega kamera prikazuje vsebino), perspektivna projekcija, začetek in konec prikazovanja vsebine (angl. near/far clipping planes), uporaba tehnik HDR (uravnoteženo prikazovanje visokega razpona barv), MSAA (glajenje robov v sceni) in izločanje zakritih ploskev, ki trenutno niso v projekciji kamere (angl. occlusion culling). Prav tako imamo na kameri pripeto skripto, ki nam omogoča: njeno translacijo po sceni, izbiro željenega objekta, orbitacijo in možnosti približevanja in oddaljevanja okrog tega.

Poglavje 5

Proceduralno generiranje mesta

V pričujočem poglavju bomo opisali postopke in uporabljene principe potrebne, za proceduralno generiranje mesta, in predstavili končni rezultat. Postopek PCG gradnje mesta bomo ločili na tri dele. V prvem bomo definirali, s kakšno podatkovno strukturo smo mesto predstavili in postavili cestno omrežje, ter kako smo definirali populacijo prebivalstva in s tem posledično območje ali pas poselitve. V drugem delu bomo predstavili, kako smo na območju mesta postavili različne tipe zgradb, izgradnjo teh in kako smo izven mesta postavili objekte, ki so ponazarjali vegetacijo mestnega okoliša. V tretjem delu bomo na kratko predstavili način izgradnje obrobja mesta.

5.1 Gradnja mesta

Mesto je zgrajeno iz pravokotnih površin, blokov (kvadrat) in cest. Znotraj mesta oziroma, povsod kjer je populacija prisotna, definiramo blok konstantne velikosti, ki mu pravimo urbani blok. Okrog vsakega urbanega bloka poteka cesta. Podobno velja za območja, kjer populacija prebivalstva ni prisotna, na teh območjih so bloki imenovani polja narave. Razlika je v tem, da zaradi postopkov delovanja algoritma, polja narave niso nujno enakih velikosti. Natančno delovanje si bomo ogledali v nadaljevanju.
Vse interaktivne parametre logike, ki definirajo grajenje mesta kot celote,

drži objekt, katerega bomo poimenovali „upravljalec mesta“. Na tej točki omenimo njegove glavne parametre:

- Globina mesta (število),
- Velikost populacije (število),
- Meja deleža populacije za gradnjo (odstotek),

Velikost mesta je definirana s parametrom globine. Velikost mesta oziroma velikost polja narave na dani globini se izračuna po sledeči formuli:

$$število\_blokov = 2^{maksimalna\_globina}/2^{trenutna\_globina}$$

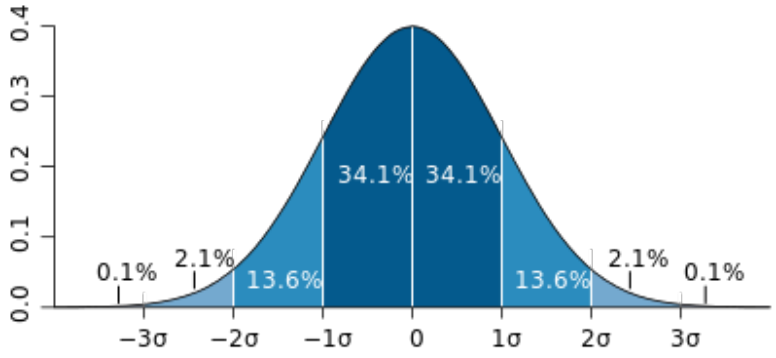
$$število\_cest = število\_blokov - 1$$

$$dolžina\_bloka = število\_blokov * širina\_bloka + število\_cest * širina\_ceste$$

Pri tem omenimo, da za spremenljivko *širina_bloka* vzamemo širino urbanega bloka, kateri predstavlja najmanjši blok mesta. Slednja enačba je potrebna zato, da širina mesta vsebuje definirano število cest, ki jih moramo zaradi delitve, opisane v nadaljevanju, všteti pri računanju širine obsega mesta ali poljubnega polja narave.
Nadalje ima uporabnik možnost izbire velikosti populacije danega mesta, s katerim definira gostoto porazdelitve populacije v mestu.
Ob zagonu programa ima uporabnik tudi možnost opredelitve željene meje, ki jo določa delež populacije prebivalstva v odsotkih, ki ponazarja spodnjo mejo deleža prebivalstva, potrebnega za izgradnjo urbanega bloka. V kolikor je delež prebivalstva trenutnega območja mesta (vozlišča podatkovne strukture opisane nižje) nižji od te meje, potem bo algoritem znotraj danega segmenta postavil polje narave, katerega velikost je odvisna od globine, na kateri se nahaja.

Prvi korak algoritma za izgradnjo mesta je porazdelitev populacije čez mesto željene velikosti. Na ta način bo algoritem kasneje pri izgradnji željenih

blokov računal deleže prebivalstva danega nivoja in temu primerno postavil blok željenega tipa in velikosti. Populacijo smo predstavili kot nabor točk v dvodimenzionalnem kartezičnem koordinatnem sistemu. Za porazdelitev populacije čez mesto smo uporabili Gaussovo ali normalno porazdelitev, oziroma natančneje standardno normalno porazdelitev z aritmetično sredino nič in odklonom velikosti ena. Za slednjo obliko porazdelitve populacije smo se odločili zato, da imajo mesta v širšem pogledu najvišji delež gostote prebivalstva ravno v njihovih središčih. Standardna normalna porazdelitev predpostavlja verjetnostno porazdelitev danega vzorca poljubne velikosti (priporočeno je, da je velikost tega večja od števila trideset). Slika 5.1 prikazuje verjetnostno porazdelitev zvonaste oblike standardne normalne porazdelitve.



Slika 5.1: Verjetnostna porazdelitev standardne normalne porazdelitve

Za uporabo te porazdelitve moramo zagotoviti premik aritmetične sredine v središče mesta in za odklon definirati šestino dolžine mesta (polovico dolžine mesta razdelimo na tri enote, ki predstavljajo standardne odklone verjetnostne standardne porazdelitve). Nadalje lahko predpostavimo, da bo taka porazdelitev postavila približno: 66 odstotkov prebivalstva v središče mesta, z odklonom ene enote v obe smeri, 28 odstotkov prebivalstva v drugi interval obeh polovic, 4 odstotke prebivalstva v tretji interval obeh polovic in preostanek, ki tvori manj kot pol odstotka, v preostali interval, ki na obeh polovicah

osi sega v neskončnost. Opisano porazdelitev smo uporabili dvakrat, prvič za definiranje koordinate x osi in drugič za definiranje koordinate y osi dane točke. Postopek smo ponovili za celoten vzorec populacije in s tem pridobili populacijo, ki nam služi kot vhodni parameter algoritma PCG.

Za odločanje, na katerem območju mesta naj se generirata željeni tip in velikost bloka, smo uporabili tako imenovano drevesno podatkovno strukturo „quadtree“, ki smo jo priredili. gre za drevesno strukturo, v kateri je vsako vozlišče definirano s štirimi otroki (v drevesnih strukturah slednji predstavljajo vgnezdena vozlišča). Je uporabna, kadar želimo učinkoviteje razdeliti neko površino. V našem primeru strukturo uporabljamo za rekurzivno delitev območja mesta na štiri dele, do predpisane globine ali v kolikor je delež populacije na dani globini prenizek. V kolikor je delež populacije danega vozlišča nižji od dovoljene meje (meja deleža populacije za gradnjo), potem algoritem danemu vozlišču predpiše naravni blok primerne velikosti. Če je globina na danem vozlišču enaka navečji dovoljeni globini, potem vozlišču predpiše urbani blok, sicer nadajuje z delitvijo. Delitev območja vedno poteka po njegovi sredini, ali povedano drugače, območje vedno razdelimo na štiri kvadrante. Vsakič, ko območje mesta razdelimo, moramo pred tem vsakemu od štirih vozlišč posredovati delež populacije, ki mu pripada. Tega izračunamo tako, da seštejemo obseg populacije, ki spada v dano območje podvozlišča (kvadranta) in ga delimo z vzorcem celotne populacije. Ko vsem štirim otrokom vozlišča porazdelimo populacijo, zaradi performančnih razlogov poskrbimo za izbris podatkovne strukture, ki hrani populacijo danega vozlišča, saj je v tem vozlišču ne potrebujemo več. Za vsako stopnjo rekurzije deljenja mesta moramo zagotoviti primerno nižjo mejo deleža populacije za gradnjo. Tako jo ob vsaki delitvi delimo na štiri.
Predstavljena koda ponazarja rekurzivno delovanje opisano zgoraj:

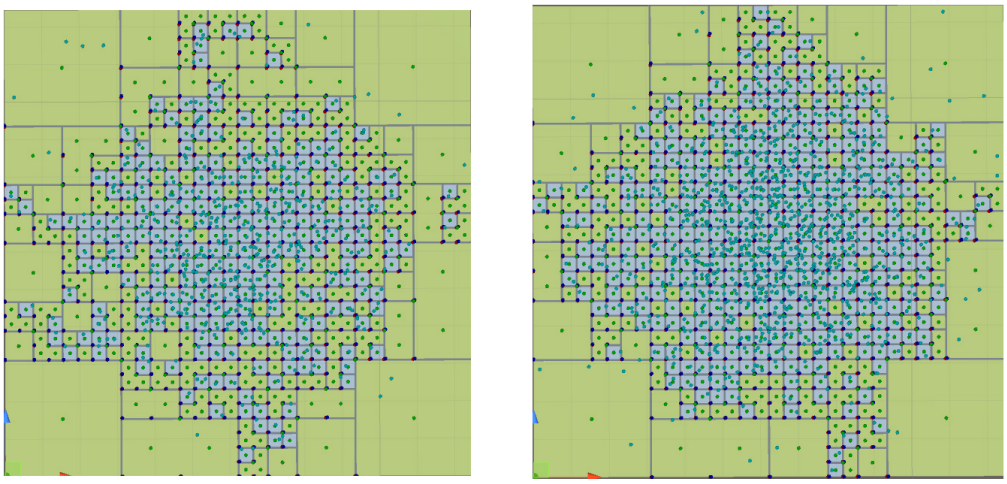
```
void Populate() {
    if (!ShouldPopulate()) {
        HasPopulation = false;
        ClearPopulation();
        return;
```

```
        }
        if (Depth < MaxAllowedDepth) {
            Subdivide();
        }
        if (Depth == MaxAllowedDepth) {
            HasPopulation = true;
            ClearPopulation();
        }
    }
```

Omenimo zastavico *HasPopulation*, ki je globalna vrednost vsakega vozlišča in ponazarja, ali dano vozlišče vsebuje zadosten delež populacije prebivalstva. Zaradi lažjega opisovanja vsebine opredelimo pozitivno (*true*) vrednost zastavice s pojmom, ki sporoča, da blok vsebuje populacijo. Razlog za to je, da vso logiko postavljanja zgradb in ostalih objektov izvajamo po izvedbi rekurzivnega deljenja, z uporabo nove rekurzije, ki iterira čez vsa novo nastala vozlišča. V kolikor bi želeli graditi zgradbe in cestno omrežje znotraj izvajanja prvotnega rekurzivnega poglabljanja, bi bili omejeni z lokalnim pogledom na mesto (zaradi izvajanja rekurzije še nimamo podatkov o še ne nastalih vozliščih). Tako pa izkoriščamo podatke vozlišč pri gradnji na globalni ravni, kar nam omogoča doseganja kvalitetnejših rezultatov. Slednji način nam za poljuben urbani blok pomaga opredeliti tip zgradb, ki ga poseljujejo, v odvisnosti od oddaljenosti bloka od središča mesta, saj bolj kot smo oddaljeni od mestnega jedra, manjšo verjetnost postavitve večnadstropnih zgradb želimo. Slednji pristop uporabljamo tudi za grajenje cestnega omrežja mesta.
Pri drugem rekurzivnem poglabljanju se prav tako poglabljamo globinsko, čez vse otroke trenutnega vozlišča, kjer s poglabljanjem začnemo v korenu drevesa. Za tem, ko rekurzija za vsako podvozlišče postavi željen tip bloka in ga naseli z željeno vsebino, se na tej točki pregledajo vsa podvozlišča in na podlagi pravil se okrog danih blokov podvozlišč dodelijo bodisi cestni pasovi bodisi pasovi narave. Za vsak blok podvozlišča pregleda, ali sosednje vozlišče trenutnega vsebuje populacijo in v tem primeru na primerni stranici bloka postavi cestni pas primerne dolžine, sicer na istem mestu postavi enako dolg pas narave, ki služi kot zapolnjevanje vrzeli, kjer cest ni. Dolžina ceste in pas narave sta privzeto enako dolga kot najmanjši blok. V kolikor je blok,

okrog katerega želimo postaviti cesto, večji, se ta skalira na primerno velikost. Sedaj nam preostane zgolj še središče vozlišča, ki ga zapolnimo bodisi s cestnim križiščem (v kolikor vsaj eno podvozlišče vsebuje populacijo) bodisi z poljem narave, zavoljo zapolnitve nastale vrzeli. Spodnja koda dobro prikazuje zaporedje opisanega postopka gradnje blokov, njihove vsebine ter gradnje cest ali pasov narave okrog njih.

```
void ConstructCity(QuadTree cell) {
    if (cell == null) {
        return;
    }
    if (!cell.HasPopulation) {
        GameObject block = PlaceBlock(NatureBlock, cell.BlockBounds);
        NatureBlockController controller;
        controller = block.GetComponent<NatureBlockController>();
        controller.PopulateNatureBlock(cell.BlockBounds, cell.Depth);
        return;
    }
    if (cell.Depth == MaxAllowedDepth) {
        GameObject block = PlaceBlock(UrbanBlock, cell.BlockBounds);
        UrbanBlockController controller;
        controller = block.GetComponent<UrbanBlockController>();
        controller.PopulateBlock(cell.BlockBounds, cell.PopulationDensity);
        return;
    }
    foreach (QuadTree childCell in cell.Cells) {
        ConstructCity(childCell);
    }
    PlaceRoads(cell, CalculateStripsInformation(cell));
}
```



Slika 5.2: Pri levi izgradnji, smo za gradnjo mesta uporabili vzorec populacije prebivalstva velikosti 500, kjer smo pri drugi za to vrednost vzeli število 1000. Obe mesti imata definirano globino vrednosti 5. Točke predstavljene s cian barvo predstavljajo porazdeljeno populacijo prebivalstva.

5.2 Postavitev vegetacije

Poselitev vegetacije, kot smo že v prejšnjem poglavju omenili, poteka na poljih narave, kjer populacije prebivalstva ni. Za objekte, ki predstavljajo vegetacijo, smo si izbrali tri vrste dreves, in sicer eno zimzeleno ter dve listnati. Populiranje vegetacije polja poljubne velikosti smo izvedli z izbiro perlinovega šuma (angl. Perlin noise) [19], ki je ena izmed zelo popularnih izbir proceduralne gradnje vsebine.

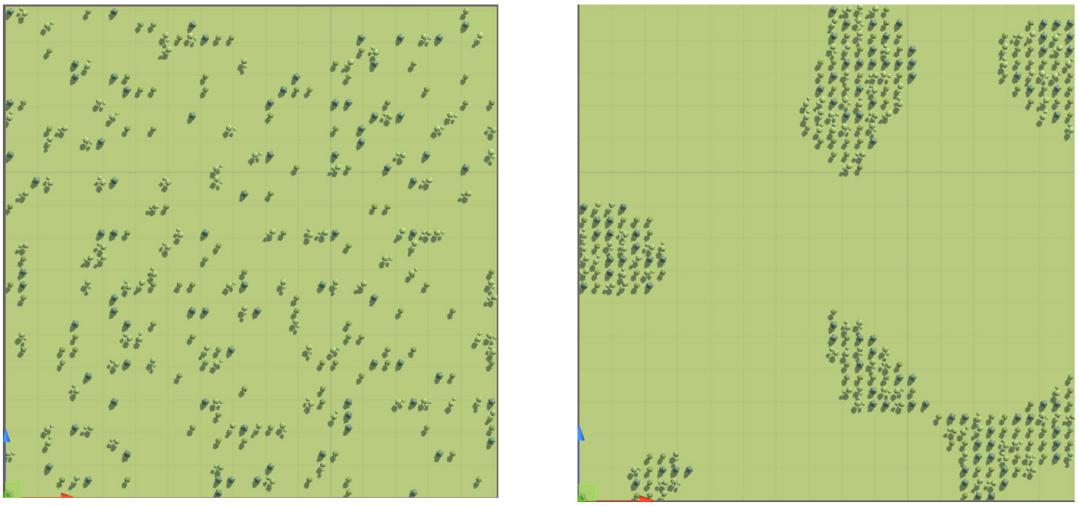
Za razliko od naključnih generatorjev, katerih porazdelitev vsebine izgleda

Slika 5.3: Perlinov šum

nenaravno, se perlinov šum opira na izdajanje kontrolirano naključne vrednosti. Ob opazovanju vrednosti vzorca perlinovega šuma opazimo, da so njegove vrednosti predstavljene kot skupek valov, ki se gradientno višajo ter nižajo čez vzorec. Perlinov šum vzame na vhodu vsako dvodimenzionalno točko in izda vrednost med nič in ena, ki je predstavljena kot ena točka njegovega vzorca. Šum bo za isto točko izdal vedno enako vrednost. Perlinov šum kot polje je neskončno, torej, če želimo v igro vključiti še spremenljivko naključja, preprosto vzamemo naključen kos dane velikosti vzorca šuma. Ob pogledu na sliko, če vzamemo za vzorec kar celotno sliko, opazimo belo-črne

gruče, ki so zgolj vrednosti med nič (bela barva) in ena (črna barva). Če za vzorec vzamemo polovico ali pa še manj, četrtino te slike, in to skaliramo na njeno prvotno velikost, opazimo, da so sedaj te gruče vrednosti mnogo večje. Temu pravimo velikost vzorca šuma.

Pri gradnji dreves smo uporabili velikost vzorca šuma števila dvajset in z njem dosegli naravno združevanje dreves, kot ga opazimo v naravi. Čez celotno polje narave smo sprva izračunali, koliko dreves lahko sodi nanj, pri tem da se drevesa med seboj ne prekrivajo. S tem smo tudi pridobili položaje potencialnih dreves, razporejenih v mrežo, raztegnjeno čez dano polje. Iterirali smo čez vsak položaj, njegove koordinate pomnožili z naključno velikim številom, ki je bilo izračunano pred iteriranjem in je enako za vsa naravna polja (s tem smo uporabili naključen vzorec šuma) in slednjo vrednost postavili na vhod perlinovega šuma, ki je izstavil vrednost na intervalu med nič in ena. V kolikor je decimalna vrednost znotraj intervala, ki se prav tako izračuna pred začetkom iteriranja in je enak za vsa naravna polja (s tem določimo naključnost gostote naše vegetacije), smo na danem mestu postavili naključno drevo iz opisanega nabora dreves, sicer je položaj ostal neposeljen.
Na tem mestu je potrebno omeniti, da smo nato vse poligonske mreže, vseh dreves polja, zaradi performančnih razlogov, združili v eno, ki je predstavljala en združen objekt. Performančni razlog, ki stoji za tem, je dejstvo, da mora igralni pogon za vsak objekt v sceni grafičnemu API-ju izdati sporočilo (angl. draw call), naj ta izriše željeni objekt. Z znižanjem števila teh izdanih klicev znatno pripomoremo k zagotavljanju zadovoljivega števila sličic prikazovanja vsebine našega projekta. V kolikor te optimizacije nismo izvedli, smo za mesto, definirano z globino pet, opazili padec sličic na sekundo s približno 75 na 5.



Slika 5.4: V obeh primerih smo za spodnjo mejo prikaza dreves uporabili vrednost 0.3. Opazimo grupiranje vrednosti v odvisnosti od velikosti vzetega vzorca.

5.3 Postavitev zgradb

Postavitev zgradb na vsak urbani otok se izvede takoj za njegovo postavitvijo v sceno. Na tej točki omenimo vse različne tipe zgradb, ki so lahko prisotne na poljubnem bloku (grafične modele slednjih smo predstavili v podpoglavju 4.1.4):

- večnadstropne zgradbe,
- večstanovanjske zgradbe,
- poslovne zgradbe,
- trgovine,
- javni zgradbi (pošta in banka),

- posebne zgradbe (univerza, bolnišnica in policijska postaja).

Posebne zgradbe so prav tako javni tip zgradb, vendar za razliko od prejšnjih, te obsegajo celotno površino urbanega bloka. Prva stvar, ki jo kontroler (algoritem, ki izvaja logiko bloka) izvede, je, da izračuna obstoj posebne zgradbe na danem bloku. Posebnim zgradbam smo za pojavitev na danem bloku dodelili verjetnost 12.5 odstotka. V kolikor je verjetnost nižja od specificirane, bo kontroler namesto postavitve posebne zgradbe začel z izvajanjem izračunavanja verjetnosti določenega tipa zgradb, ki so lahko prisotne na danem bloku. Upravljalec mesta vsebuje dodatne parametre, ki jih prej nismo omenili. To so uteži, ki deljene z njihovo kumulativno vsoto, predstavljajo verjetnosti pojavitve dane zgradbe. Upravljalec mesta tako vsebuje štiri take uteži, kot utež pojavitve večstanovanjske zgradbe, poslovne zgradbe, trgovine in javne zgradbe. Bralec opazi, da večnadstropne zgradbe nimajo specificirane uteži pojavitve, saj se verjetnost slednje izračuna ločeno. Takim zgradbam izračunamo verjetnost pojavitve v odvisnosti od trenutnega deleža populacije, ki je definiran na danem bloku. Natančneje lahko opišemo verjetnost večnadstropne zgradbe v obliki padajoče linearne funkcije, ki bo na bloku z maksimalnim deležem poselitve prebivalstva zajela stoodstotno verjetnost poselitve in na bloku, kjer je delež populacije prebivalstva ravno še sprejemljiv za izgradnjo urbanega bloka, zajela verjetnost nič. Prav tako omejujemo pas seganja večnadstropnih zgradb s še zadnjim neomenjenim interaktivnim parametrom, ki vpliva na logiko PCG. Slednji je definiran kot delež v odstotkih in predstavlja površino (s središčem v centru mesta), na kateri je dovoljeno grajenje teh stavb. Parameter vrednosti ena definira možnost poselitve vsakega urbanega bloka s tem tipom zgradb.
Računanje verjetnosti ponazorimo s kodo:

```
float interval = maxPopulationDensity - currentPopulationDensity;
float p = maxPopulationDensity - currentPopulationDensity;
return 1 - p / interval;
```

Preostanek verjetnosti je enak $1 - verjetnost\_večnadstropne\_zgradbe$ in ga uporabimo kot osnovo za porazdelitev verjetnosti, izračunanih iz uteži preostalih tipov zgradb. Za lažje razumevanje podajmo primer. Predpostavimo

verjetnost 0.4 pojavitve večnadstropne zgradbe. Definirajmo naslednje uteži preostalih tipov zgradb v zaporedju, navedenem na začetku podpoglavja (z izjemo posebnih zgradb), kot: 3, 1.5, 1.5 in 0. Zaradi uporabe uteži so lahko večje od ena in se ne seštevajo v ena, saj se ob njihovi pretvorbi v zapis verjetnosti normalizirajo. Dobimo slednje verjetnosti: večnadstropna zgradba (p = 0.4), večstanovanjska zgradba (p = 0.3), poslovna zgradba (p = 0.15), trgovina (p = 0.15), javna zgradba (p = 0). Sedaj, ko imamo definirane verjetnosti pojavitve vsakega tipa zgradbe, opišimo postopek postavitve le teh na blok. Pri postavljanju zgradb na blok smo se odločili za tako imenovano obliko obsega, kjer se zgradbe postavljajo na robove danega bloka. Pri tem, ko rečemo, da se postavljajo na robove bloka, ne mislimo dobesedno na njegov rob, ampak na robove obsega bloka, ki je nekoliko pomaknjen v njegovo notranjost, saj s tem zunanji del bloka ponazorimo kot pločnik. Preden kontroler prične s postavitvijo zgradb na blok, jih mora najprej pridobiti. Definiramo predpisano število zgradb, ki jih blok ustvari vnaprej, zaradi efektivnejše postavitve le teh. Pridobljene zgrajene zgradbe so zaradi performančnih razlogov urejene po njihovih padajočih širinah. Na tem mestu omenimo, da se poslovne zgradbe in trgovine nadalje delijo na tako imenovane „vogalne“ ali „corner“ zgradbe. Te zgradbe so bile ustvarjene zaradi lepše zapolnitve vogalov bloka. Ob pridobitvi željenih zgrajenih zgradb, kontroler sprva prične s postavitvijo vogalnih zgradb na vogale bloka, kjer znotraj pridobljene liste poišče vogalne zgradbe in jih postavi na željena mesta. V kolikor je vogalnih zgradb manj kot vogalov bloka, izbere naključno zgradbo in jo postavi na dano mesto. Naj omenimo, da logika, ki pridobi željene zgradbe, onemogoča, da bi bile znotraj lista več kot štiri vogalne, saj si ne želimo takih zgradb nikjer drugje kot zgolj na vogalih bloka. Ko so vogalne zgradbe postavljene, se izračunajo začetne točke, na katerih bo kontroler pričel s postavitvijo zgradb danega roba. Kontroler pri postavitvi zgradb sledi naslednjim pravilom:

1. Izberi naključno zgradbo.

2. Poglej, ali je med trenutnim položajem in naslednjo vogalno zgradbo

prostor za umestitev izbrane zgradbe.

3. V kolikor je prostor, postavi izbrano zgradbo in skoči na korak 7.

4. V kolikor je prostor za umestitev izbrane zgradbe manjši od širine izbrane zgradbe, poišči največjo zgradbo, ki se še lahko postavi v dani prostor.

5. V kolikor je prostor, postavi izbrano zgradbo in skoči na korak 7.

6. V kolikor ni prostora, poizkusi zapolniti prostor z naključnim „mašilom“ (za njihovo predstavitev glej sliko 5.3), v kolikor tudi zanj ni prostora, poizkusi poiskati še največje tako, ki lahko zapolni prostor in izstopi iz zanke.

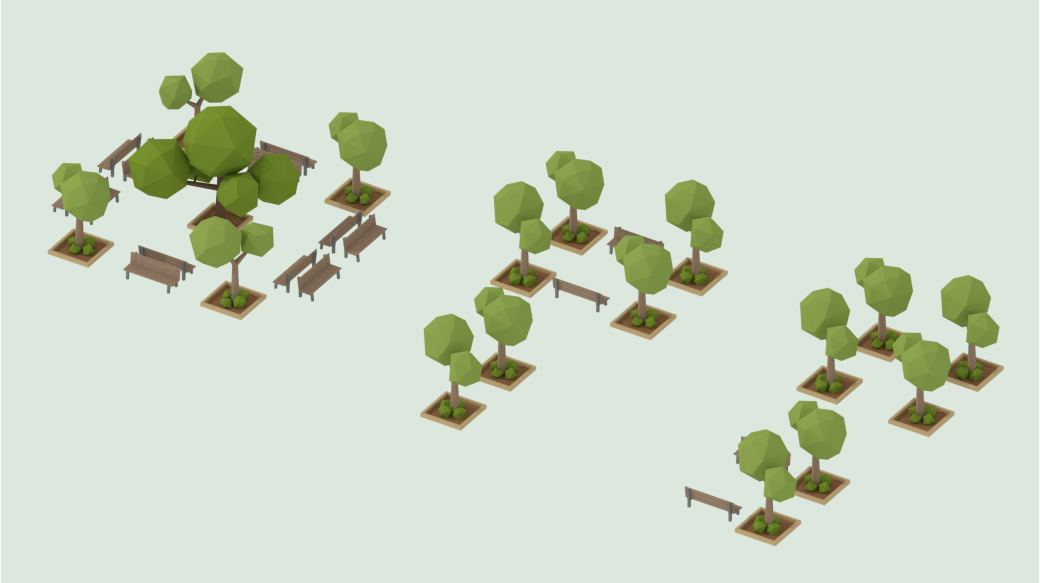
7. Premakni izhodiščno točko, ki predstavlja trenutni položaj, vzdolž roba za širino zgradbe in dolžino, ki predstavlja velikost presledka med stavbami.

Pod točko št. 2 se ob prvem prihodu v zanko preveri dolžina od trenutnega položaja do končnega, s tehniko metanja žarka (angl. raycasting), zgolj za pridobitev referenčne točke, ki predstavlja zadnjo stranico vogalne zgradbe. Pri vsakem naslednjem obhodu aritmetično izračunamo dolžino od trenutnega položaja do referenčne točke. Zaradi take oblike postavljanja zgradb je sredina bloka prazna, slednjo zapolnimo z vnaprej zgrajenim skupkom objektov, ki predstavljajo park obsegajočih zgradb.

Do te točke smo navedli vse potrebne informacije, ki opisujejo izgradnjo mesta, razen postopka izgradnje posameznega tipa zgradbe. Vsaka zgradba se zgradi ob klicu, ko kontroler zahteva pridobitev željenih zgrajenih zgradb za potrebe postavljanja le teh. Vsaka zgradba je minimalno zgrajena iz spodnjega dela (angl. bot), srednjega dela (angl. mid) in zgornjega dela (angl. top). Vsak nivo zgradbe ima definiranih več različnih oblik, ki smo jih oblikovali v okolju Blender. Poljubno je lahko dopolnjena z vegetacijo, ki

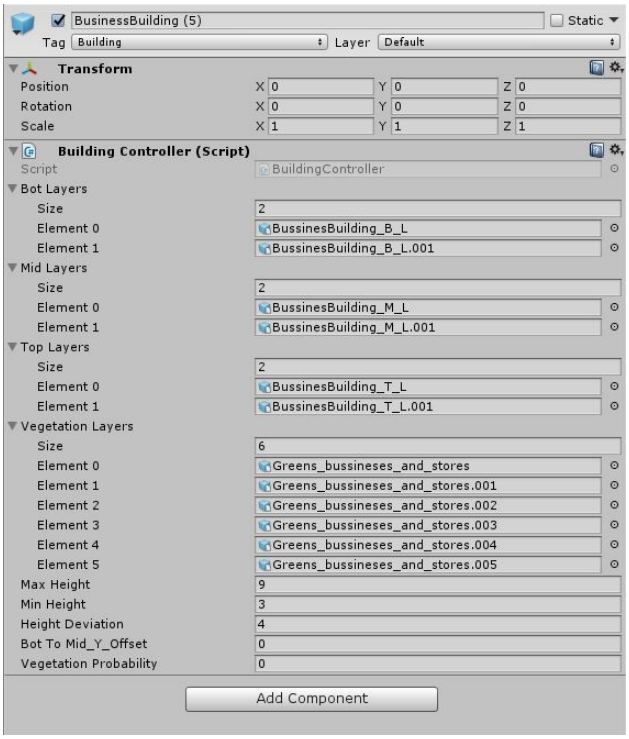
Slika 5.5: Na sliki opazmo dva primera, pri katerih je algoritem dobro zapolnil preostanek prostora. V prvem je zanj našel primerno široko zgradbo, pri drugem pa je bila vrzel premajhna za umestitev zgradbe, zato jo je zapolnil z mašilom.

obdaja dano stavbo, odvisno od verjetnosti, ki definira njen obstoj. Zopet je nivo vegetacije lahko predstavljen z večjim številom različnih oblik. Slika 5.7 prikazuje vse parametre, ki so potrebni za izgradnjo posamezne zgradbe. Vsaka zgradba je določenega tipa in določene barve, ki ju definiramo. Poleg omenjenih parametrov je vsaka zgradba omejena s predpisano minimalno, maksimalno višino in višino odklanjanja. Slednji parametri se uporabljajo pri določitvi višine zgradbe. Višina zgradbe je pogojena na podoben način kot verjetnost pojavitve večnadstropne zgradbe na danem bloku. Njena višina je definirana s padajočo funkcijo, katere os y je omejena z njeno minimalno in maksimalno višino, os x pa je omejena z maksimalnim deležem prebivalstva in minimalnim deležem prebivalstva mesta. Zgradba v središču mesta bo definirana z njeno maksimalno višino in zgradba na obrobjih mesta bo definirana z njeno minimalno višino. Slednje pomeni, da bodo vse zgradbe na določenem



Slika 5.6: Na desni strani slike so ponazorjeni skupki objektov, ki ponazarjajo mašila. Objekt na levi predstavlja skupek objektov, ki se zaradi zapolnjevanja praznega prostora, postavi na središče vsakega urbanega bloka.

bloku identičnih višin, v kolikor sta njihova minimalna in maksimalna velikost enaki. Zaradi slednjega nad izračunano višino izvedemo standardno normalno porazdelitev, ki za aritmetično vsebino vzame izračunano višino zgradbe ter parameter višine odklanjanja za maksimalni možni odklon. V kolikor je nova višina zgradbe manjša od minimalne predpisane, se predpostavi minimalna vrednost. Zadnje ne velja, v kolikor nova višina sega preko maksimalne dovoljene. Nova višina, od katere odštejemo vrednost števila dve, (predstavlja zgornji in spodnji nivo zgradbe) nam predstavlja število srednjih slojev zgradbe iz, katerih je zgradba sestavljena. V okolju Blender smo za pivote vseh slojev morali zagotoviti, da so poravnani po navpični osi, saj vse sloje zgolj zlagamo drugega na drugega. Kot pri populaciji polj narave, tu uporabimo isti koncept združevanja vseh poligonskih mrež nivojev zgradbe.



Slika 5.7: Parametri zgradbe, ki definirajo njen izgled in višino.

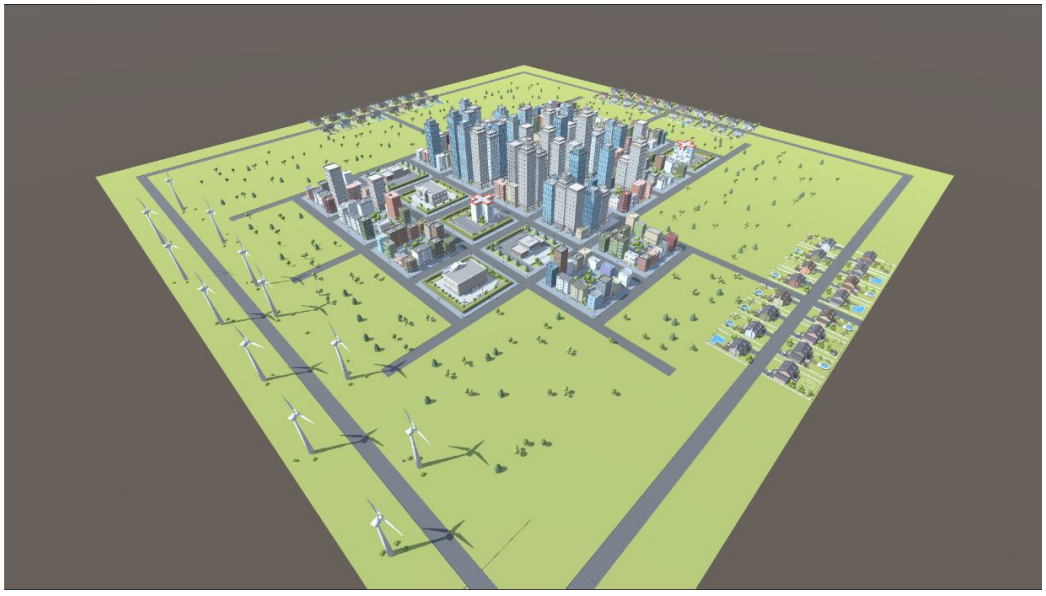
5.4 Obrobje mesta

Obrobje mesta je ponazorjeno na vseh robovih mesta, kjer so na treh robovih upodobljene soseske hiš, ki predstavljajo naselja, ločena od mesta z zelenimi pasovi (predstavljena kot polja narave med naseljem in mestom), kot to opazimo pri večini znanih angleških mest (npr. London, Oxford, Cambridge, itd...). Zadnji rob je predstavljen z gručami postavljenih veternic. Gradnja sosesk je zelo podobna gradnji urbanih zgradb, kjer namesto treh ali štirih nivojev uporabljamo zgolj dva. Prvi nivo predstavlja hišo, kjer drugi predstavlja njen vrt, oba nivoja sta obvezna pri izgradnji hiše. Blok, na katerem so postavljene soseske in vetrnice, je narejen zaradi potrebe po prikazu obrobja mesta. Tega sestavlja blok narave, ki po sredini vsebuje cesto, ta pa se rapotega do robov bloka. Dolžina sosesk, kot tudi velikost območja postavitve vetrnic, se izračuna za vsako poljubno velikost mesta, kot tudi obrobje mesta

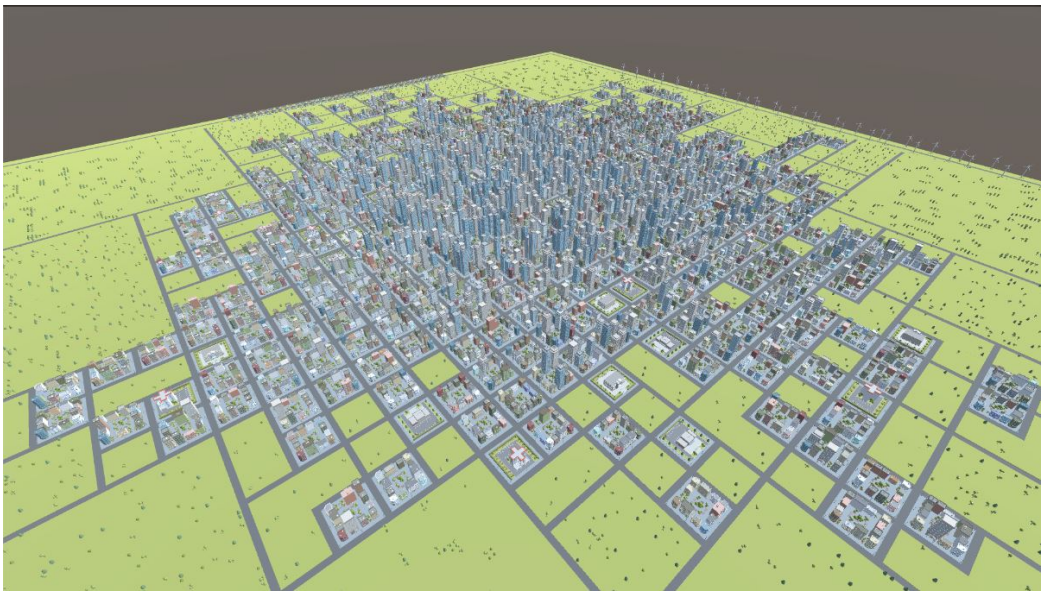
(naselja imajo različno dolžino), na podlagi izbire naključne dolžine definiranega intervala. Za dodeljeno dolžino se nadalje izračuna število objektov, ki jih nato postavimo s pred definirano oddaljenostjo drug od drugega. Zadnje velja tako za hiše, ki sestavljajo naselja, kot tudi za gruče vetrnic na dani površini.

5.5 Prikaz rezultatov

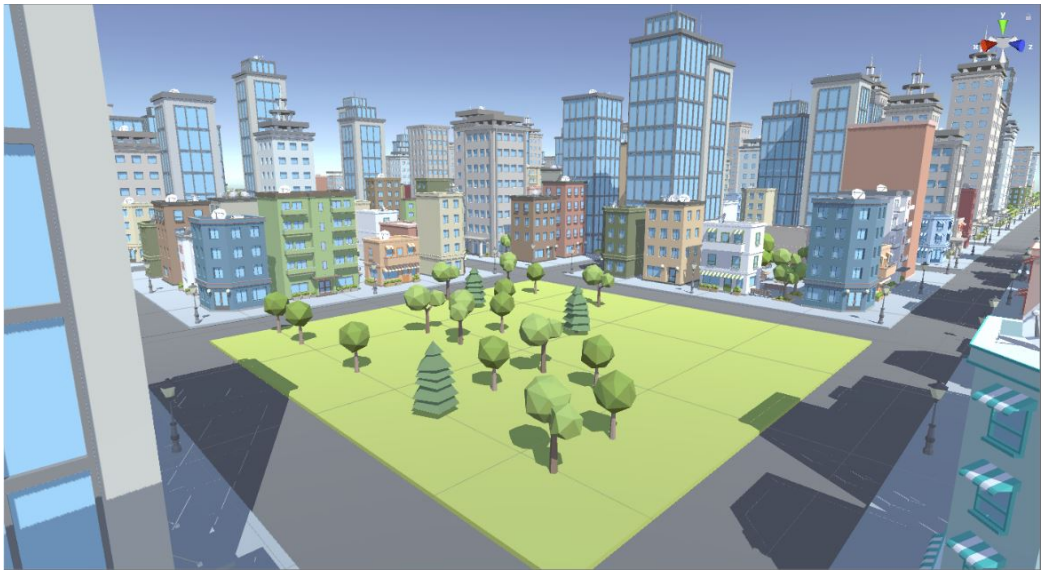
Predstavljamo rezultate kot skupek slik, ki ponazarjajo zgrajena mesta in njihove vsebine.



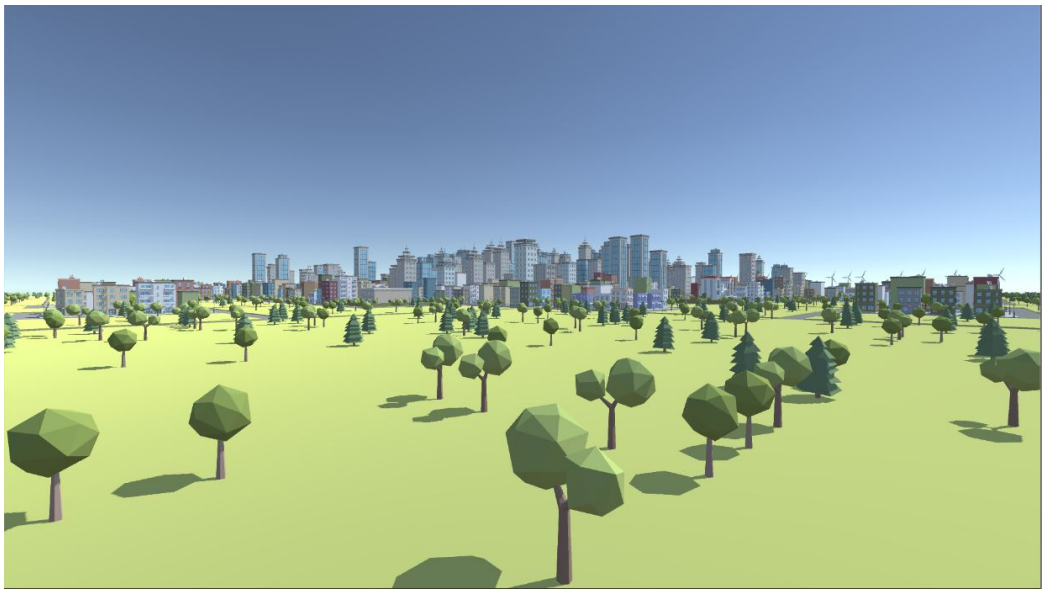
Slika 5.8: Mesto velikosti globine tri



Slika 5.9: Mesto velikosti globine pet



Slika 5.10: Pogled na polje narave, ki jo obdaja urbana infrastruktura



Slika 5.11: Pogled v mestno središče

Slika 5.12: Prikaz naselja s pogledom na mesto in vetrnice v ozadju

Slika 5.13: Slika mestnega obrobja

Slika 5.14: Pogled na univerzo in mestno vpadnico

Poglavje 6

Sklepne ugotovitve

Znotraj diplomskega dela smo najprej pregledali področje interaktive računalniške grafike in stila low poly, nadaljevali smo s predstavitvijo proceduralnega generiranja vsebine in zaključili z izgradnjo programske opreme, v kateri smo osvojeno znanje združili ter s postopki PCG zgradili poljubno mesto, čigar grafična vsebina je bila oblikovana v stilu low poly. Pri gradnji mesta smo opazili nekaj izboljšav, ki bi lahko z uporabo naprednejših algoritmov predstavile še bogatejšo vsebino. Naše mesto se pri izgradnji opira zgolj na obliko standardne normalne porazdelitve. Z uporabo večjega nabora različnih vrst porazdelitev in njihovih kombinacij bi lahko vplivali na diverziteto oblike mesta. Z uporabo naprednejših algoritmov za izgradnjo cestnih omrežij bi se znebili prisotnosti slepih poti. Razvita programska oprema bi lahko povečala raznolikost statičnih vsebin v igrah.

Literatura

[1] 3d object representations. http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/meshes/polygon_meshes.html.

[2] Blender. https://www.blender.org/.

[3] Blender - Documentation. https://docs.blender.org/manual/en/dev/interface/window_system/introduction.html.

[4] Blender 3D view, header - Documentation. https://docs.blender.org/manual/en/dev/editors/3dview/introduction.html.

[5] Blender API - Documentation. https://docs.blender.org/api/current/.

[6] Computer graphics. https://en.wikipedia.org/wiki/Computer_graphics.

[7] Euler rule. https://www.ics.uci.edu/~eppstein/junkyard/euler/.

[8] Game Developers Conference 2017 - Low Poly Modeling: Style Through Economy by Ethan Redd. https://www.youtube.com/watch?v=H1oNuKChsdU&t=218.

[9] Game Developers Conference 2017 - Practical Procedural Generation for Everyone by Kate Compton. https://www.youtube.com/watch?v=WumyfLEa6bU.

[10] Low poly. https://en.wikipedia.org/wiki/Low_poly.

[11] No Mans Sky. https://en.wikipedia.org/wiki/No_Man%27s_Sky.

[12] Oscurart Tools addon. https://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/3D_interaction/Oscurart_Tools.

[13] Polygon mesh. https://en.wikipedia.org/wiki/Polygon_mesh.

[14] Predavanja doc. dr. Matije Marolta pri predmetu Računalniška grafika in tehnologija iger. https://ucilnica.fri.uni-lj.si/pluginfile.php/50161/mod_resource/content/0/31%20Predstavitve%20predmetov%20-%20deljene%20ploskve.pdf.

[15] Procedural content generation. https://en.wikipedia.org/wiki/Procedural_generation.

[16] Unity. https://www.python.org/.

[17] Unity. https://unity3d.com/.

[18] Unity. https://docs.unity3d.com/Manual/LearningtheInterface.html.

[19] Unity. https://en.wikipedia.org/wiki/Perlin_noise.

[20] Unity plans. https://store.unity.com/.

[21] Bjarki Gulaugsson. Procedural content generation. 2006.

[22] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer, 2016.